PPP—Point-to-Point Protocol

To improve the situation, the IETF set up a group to devise a data link protocol for point-to-point lines that solved all these problems and that could become an official Internet Standard. This work culminated in **PPP (Point-to-Point Protocol)**, which is defined in RFC 1661 and further elaborated on in several other RFCs (e.g., RFCs 1662 and 1663). PPP handles error detection, supports multiple protocols, allows IP addresses to be negotiated at connection time, permits authentication, and has many other improvements over SLIP. While many Internet service providers still support both SLIP and PPP, the future clearly lies with PPP, not only for dial-up lines, but also for leased router-router lines.

PPP provides three things:

1. A framing method that unambiguously delineates the end of one frame and the start of the next one. The frame format also handles error detection.
2. A link control protocol for bringing lines up, testing them, negotiating options, and bringing them down again gracefully when they are no longer needed. This protocol is called **LCP (Link Control Protocol)**.
3. A way to negotiate network-layer options in a way that is independent of the network layer protocol to be used. The method chosen is to have a different **NCP (Network Control Protocol)** for each network layer supported.

To see how these pieces fit together, let us consider the typical scenario of a home user calling up an Internet service provider to make a home PC a temporary Internet host. The PC first calls the provider's router via a modem. After the router's modem has answered the phone and established a physical connection, the PC sends the router a series of LCP packets in the payload field of one or more PPP frames. These packets, and their responses, select the PPP parameters to be used.

Once these have been agreed upon, a series of NCP packets are sent to configure the network layer. Typically, the PC wants to run a TCP/IP protocol stack, so it needs an IP address. There are not enough IP addresses to go around, so normally each Internet provider gets a block of them and then dynamically assigns one to each newly attached PC for the duration of its login session. If a provider owns *n* IP addresses, it can have up to *n* machines logged in simultaneously, but its total customer base may be many times that. The NCP for IP is used to do the IP address assignment.

At this point, the PC is now an Internet host and can send and receive IP packets, just as hardwired hosts can. When the user is finished, NCP is used to tear down the network layer connection and free up the IP address. Then LCP is used to shut down the data link layer connection. Finally, the computer tells the modem to hang up the phone, releasing the physical layer connection.

The PPP frame format was chosen to closely resemble the HDLC frame format, since there was no reason to reinvent the wheel. The major difference between PPP and HDLC is that the former is character oriented rather than bit oriented. In particular, PPP, like, SLIP, uses character stuffing on dial-up modem lines, so all frames are an integral number of bytes. It is not possible to send a frame consisting of 30.25 bytes, as it is with HDLC. Not only can PPP frames be sent over dial-up telephone lines, but they can also be sent over SONET or true bit-oriented HDLC lines (e.g., for router-router connections). The PPP frame format is shown in Fig. 3-27.

Bytes	1	1	1	1 or 2	Variable	2 or 4	1
	Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload	Checksum	Flag 01111110

Fig. 3-27. The PPP full frame format for unnumbered mode operation.

All PPP frames begin with the standard HDLC flag byte (01111110), which is character stuffed if it occurs within the payload field. Next comes the *Address* field, which is always set to the binary value 11111111 to indicate that all stations are to accept the frame. Using this value avoids the issue of having to assign data link addresses.

The *Address* field is followed by the *Control* field, the default value of which is 00000011. This value indicates an unnumbered frame. In other words, PPP does not provide reliable transmission using sequence numbers and acknowledgements as the default. In noisy environments, such as wireless networks, reliable transmission using numbered mode can be used. The exact details are defined in RFC 1663.

Since the *Address* and *Control* fields are always constant in the default configuration, LCP provides the necessary mechanism for the two parties to negotiate an option to just omit them altogether and save 2 bytes per frame.

The fourth PPP field is the *Protocol* field. Its job is to tell what kind of packet is in the *Payload* field. Codes are defined for LCP, NCP, IP, IPX, AppleTalk, and other protocols. Protocols starting with a 0 bit are network layer protocols such as IP, IPX, OSI CLNP, XNS. Those starting with a 1 bit are used to negotiate other protocols. These include LCP and a different NCP for each network layer protocol supported. The default size of the *Protocol* field is 2 bytes, but it can be negotiated down to 1 byte using LCP.

The *Payload* field is variable length, up to some negotiated maximum. If the length is not negotiated using LCP during line setup, a default length of 1500 bytes is used. Padding may follow the payload if need be.

After the *Payload* field comes the *Checksum* field, which is normally 2 bytes, but a 4-byte checksum can be negotiated.

In summary, PPP is a multiprotocol framing mechanism suitable for use over modems, HDLC bit-serial lines, SONET, and other physical layers. It supports error detection, option negotiation, header compression, and optionally, reliable transmission using HDLC framing.

Let us now turn from the PPP frame format to the way lines are brought up and down. The (simplified) diagram of Fig. 3-28 shows the phases that a line goes through when it is brought up, used, and taken down again. This sequence applies both to modem connections and to router-router connections.

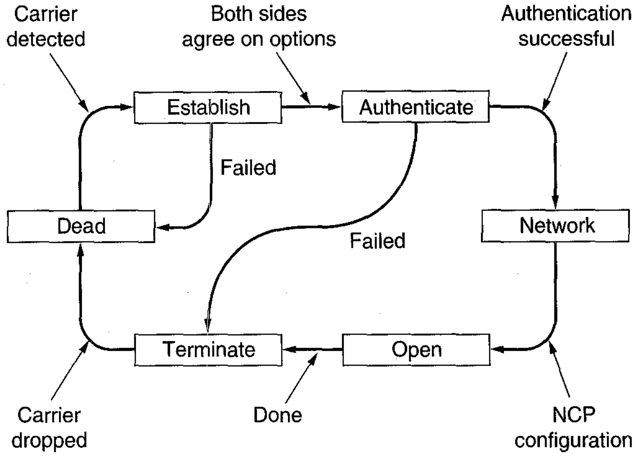


Fig. 3-28. A simplified phase diagram for bringing a line up and down.

When the line is *DEAD*, no physical layer carrier is present and no physical layer connection exists. After physical connection is established, the line moves to *ESTABLISHED*. At that point LCP option negotiation begins, which, if successful, leads to *AUTHENTICATE*. Now the two parties can check on each other's identities, if desired. When the *NETWORK* phase is entered, the appropriate NCP protocol is invoked to configure the network layer. If the configuration is successful, *OPEN* is reached and data transport can take place. When data transport is finished, the line moves into the *TERMINATE* phase, and from there, back to *DEAD* when the carrier is dropped.

LCP is used to negotiate data link protocol options during the *ESTABLISH* phase. The LCP protocol is not actually concerned with the options themselves, but with the mechanism for negotiation. It provides a way for the initiating process to make a proposal and for the responding process to accept or reject it, in whole or in part. It also provides a way for the two processes to test the line quality, to see if they consider it good enough to set up a connection. Finally, the LCP protocol also allows lines to be taken down when they are no longer needed.

Eleven types of LCP packets are defined in RFC 1661. These are listed in Fig. 3-29. The four *Configure-* types allow the initiator (I) to propose option values and the responder (R) to accept or reject them. In the latter case, the responder can make an alternative proposal or announce that it is not willing to negotiate certain options at all. The options being negotiated and their proposed values are part of the LCP packets.

Name	Direction	Description
Configure-request	I → R	List of proposed options and values
Configure-ack	I ← R	All options are accepted
Configure-nak	I ← R	Some options are not accepted
Configure-reject	I ← R	Some options are not negotiable
Terminate-request	I → R	Request to shut the line down
Terminate-ack	I ← R	OK, line shut down
Code-reject	I ← R	Unknown request received
Protocol-reject	I ← R	Unknown protocol requested
Echo-request	I → R	Please send this frame back
Echo-reply	I ← R	Here is the frame back
Discard-request	I → R	Just discard this frame (for testing)

Fig. 3-29. The LCP packet types.

The *Terminate-* codes are used to shut a line down when it is no longer needed. The *Code-reject* and *Protocol-reject* codes are used by the responder to indicate that it got something that it does not understand. This situation could mean that an undetected transmission error has occurred, but more likely it means that the initiator and responder are running different versions of the LCP protocol. The *Echo-* types are used to test the line quality. Finally, *Discard-request* is used for debugging. If either end is having trouble getting bits onto the wire, the programmer can use this type for testing. If it manages to get through, the receiver just throws it away, rather than taking some other action, which might confuse the person doing the testing.

The options that can be negotiated include setting the maximum payload size for data frames, enabling authentication and choosing a protocol to use, enabling line quality monitoring during normal operation, and selecting various header compression options.

There is little to say about the NCP protocols in a general way. Each one is specific to some network layer protocol and allows configuration requests to be made that are specific to that protocol. For IP, for example, dynamic address assignment is the most important possibility.

3.6.3. The Data Link Layer in ATM

It is now time to begin our journey up through the ATM protocol layers of Fig. 1-30. The ATM physical layer covers roughly the OSI physical and data link layers, with the physical medium dependent sublayer being functionally like the OSI physical layer and the transmission convergence (TC) sublayer having data link functionality. There are no physical layer characteristics specific to ATM. Instead, ATM cells are carried by SONET, FDDI, and other transmission systems. Therefore we will concentrate here on the data link functionality of the TC sublayer, but we will discuss some aspects of the interface with the lower sublayer later on.

When an application program produces a message to be sent, that message works its way down the ATM protocol stack, having headers and trailers added and undergoing segmentation into cells. Eventually, the cells reach the TC sublayer for transmission. Let us see what happens to them on the way out the door.

Cell Transmission

The first step is header checksumming. Each cell contains a 5-byte header consisting of 4 bytes of virtual circuit and control information followed by a 1-byte checksum. Although the contents of the header are not relevant to the TC sublayer, curious readers wishing a sneak preview should turn to Fig. 5-62. The checksum only covers the first four header bytes, not the payload field. It consists of the remainder after the 32 header bits have been divided by the polynomial $x^8 + x^2 + x + 1$. To this the constant 01010101 is added, to provide robustness in the face of headers containing mostly 0 bits.

The decision to checksum only the header was made to reduce the probability of cells being delivered incorrectly due to a header error, but to avoid paying the price of checksumming the much larger payload field. It is up to higher layers to perform this function, if they so desire. For many real-time applications, such as voice and video, losing a few bits once in a while is acceptable (although for some compression schemes, all frames are equal but some frames are more equal). Because it covers only the header, the 8-bit checksum field is called the **HEC (Header Error Control)**.

A factor that played a major role in this checksumming scheme is the fact that ATM was designed for use over fiber, and fiber is highly reliable. Furthermore, a major study of the U.S. telephone network has shown that during normal operation 99.64 percent of all errors on fiber optic lines are single-bit errors (AT&T and Bellcore, 1989). The HEC scheme corrects all single-bit errors and detects many multibit errors as well. If we assume that the probability of a single-bit error is 10^{-8}, then the probability of a cell containing a detectable multibit header error is about 10^{-13}. The probability of a cell slipping through with an undetected header error is about 10^{-20}, which means that at OC-3 speed, one bad cell header will get through every 90,000 years. Although this may sound like a long time, once the earth has, say, 1 billion ATM telephones, each used 10 percent of the time, over 1000 bad cell headers per year will go undetected.

For applications that need reliable transmission in the data link layer, Shacham and McKenney (1990) have developed a scheme in which a sequence of consecutive cells are EXCLUSIVE ORed together. The result, an entire cell, is appended to the sequence. If one cell is lost or badly garbled, it can be reconstructed from the available information.

Once the HEC has been generated and inserted into the cell header, the cell is ready for transmission. Transmission media come in two categories: asynchronous and synchronous. When an asynchronous medium is used, a cell can be sent whenever it is ready to go. No timing restrictions exist.

With a synchronous medium, cells must be transmitted according to a predefined timing pattern. If no data cell is available when needed, the TC sublayer must invent one. These are called **idle cells**.

Another kind of nondata cell is the **OAM** (**Operation And Maintenance**) cell. OAM cells are also used by the ATM switches for exchanging control and other information necessary for keeping the system running. OAM cells also have some other special functions. For example, the 155.52-Mbps OC-3 speed matches the gross data rate of SONET, but an STM-1 frame has a total of 10 columns of overhead out of 270, so the SONET payload is only 260/270 × 155.52 Mbps or 149.76 Mbps. To keep from swamping SONET, an ATM source using SONET would normally put out an OAM cell as every 27th cell, to slow the data rate down to 26/27 of 155.52 Mbps and thus match SONET exactly. The job of matching the ATM output rate to the rate of the underlying transmission system is an important task of the TC sublayer.

On the receiver's side, idle cells are processed in the TC sublayer, but OAM cells are given to the ATM layer. OAM cells are distinguished from data cells by having the first three header bytes be all zeros, something not allowed for data cells. The fourth byte describes the nature of the OAM cell.

Another important task of the TC sublayer is generating the framing information for the underlying transmission system, if any. For example, an ATM video camera might just produce a sequence of cells on the wire, but it might also produce SONET frames with the ATM cells embedded inside the SONET payload. In the latter case, the TC sublayer would generate the SONET framing and pack the ATM cells inside, not entirely a trivial business since a SONET payload does not hold an integral number of 53-byte cells.

Although the telephone companies clearly intend to use SONET as the underlying transmission system for ATM, mappings from ATM onto the payload fields of other systems have also been defined, and new ones are being worked on. In particular, mappings onto T1, T3, and FDDI also exist.

Cell Reception

On output, the job of the TC sublayer is to take a sequence of cells, add a HEC to each one, convert the result to a bit stream, and match the bit stream to the speed of the underlying physical transmission system by inserting OAM cells as filler. On input, the TC sublayer does exactly the reverse. It takes an incoming bit stream, locates the cell boundaries, verifies the headers (discarding cells with invalid headers), processes the OAM cells, and passes the data cells up to the ATM layer.

The hardest part is locating the cell boundaries in the incoming bit stream. At the bit level, a cell is just a sequence of $53 \times 8 = 424$ bits. No 01111110 flag bytes are present to mark the start and end of a cell, as they are in HDLC. In fact, there are no markers at all. How can cell boundaries be recognized under these circumstances?

In some case, the underlying physical layer provides help. With SONET, for example, cells can be aligned with the synchronous payload envelope, so the SPE pointer in the SONET header points to the start of the first full cell. However, sometimes the physical layer provides no assistance in framing. What then?

The trick is to use the HEC. As the bits come in, the TC sublayer maintains a 40-bit shift register, with bits entering on the left and exiting on the right. The TC sublayer then inspects the 40 bits to see if it is potentially a valid cell header. If it is, the rightmost 8 bits will be valid HEC over the leftmost 32 bits. If this condition does not hold, the buffer does not hold a valid cell, in which case all the bits in the buffer are shifted right one bit, causing one bit to fall off the end, and a new input bit is inserted at the left end. This process is repeated until a valid HEC is located. At that point, the cell boundary is known because the shift register contains a valid header.

The trouble with this heuristic is that the HEC is only 8 bits wide. For any given shift register, even one containing random bits, the probability of finding a valid HEC is 1/256, a moderately large value. Used by itself, this procedure would incorrectly detect cell headers far too often.

To improve the accuracy of the recognition algorithm, the finite state machine of Fig. 3-30 is used. Three states are used: *HUNT*, *PRESYNCH*, and *SYNCH*. In the *HUNT* state, the TC sublayer is shifting bits into the shift registers one at a time looking for a valid HEC. As soon as one is found, the finite state machine switches to *PRESYNCH* state, meaning that it has tentatively located a cell boundary. It now shifts in the next 424 bits (53 bytes) without examining them. If its guess about the cell boundary was correct, the shift register should now contain another valid cell header, so it once again runs the HEC algorithm. If the HEC is incorrect, the TC goes back to the *HUNT* state and continues to search bit-by-bit for a header whose HEC is correct.

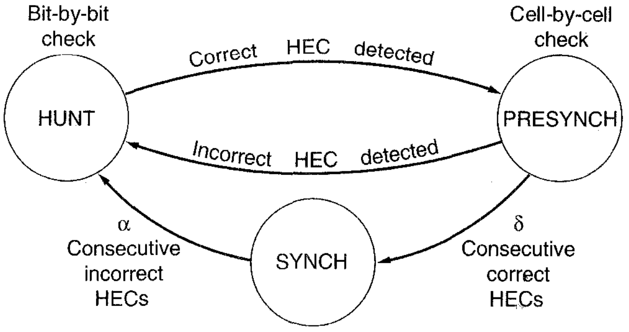


Fig. 3-30. The cell delineation heuristic.

On the other hand, if the second HEC is also correct, the TC may be onto something, so it shifts in another 424 bits and tries again. It continues inspecting headers in this fashion until it has found δ correct headers in a row, at which time it assumes that it is synchronized and moves into the *SYNCH* state to start normal operation. Note that the probability of getting into *SYNCH* state by accident with a purely random bit stream is $2^{-8\delta}$, which can be made arbitrarily small by choosing a large enough δ. The price paid for a large δ, however, is a longer time to synchronize.

In addition to resynchronizing after losing synchronization (or at startup), the TC sublayer needs a heuristic to determine when it has lost synchronization, for example after a bit has been inserted or deleted from the bit stream. It would be unwise to give up if just one HEC was incorrect, since most errors are bit inversions, not insertions or deletions. The wisest course here is just to discard the cell with the bad header and hope the next one is good. However, if α HECs in a row are bad, the TC sublayer has to conclude that it has lost synchronization and must return to the *HUNT* state.

Although unlikely, it is conceivable that a malicious user could try to spoof the TC sublayer by inserting a data pattern into the payload field of many consecutive cells that imitates the HEC algorithm. Then, if synchronization were ever lost, it might be regained in the wrong place. To make this trick much harder, the payload bits are scrambled on transmission and descrambled on reception.

Before leaving the TC sublayer, one comment is in order. The mechanism chosen for cell delineation requires the TC sublayer to understand and use the header of the ATM layer above it. Having one layer make use of the header of a higher layer is in complete violation of the basic rules of protocol engineering. The idea of having layered protocols is to make each layer be independent of the ones above it. It should be possible, for example, to change the header format of the ATM layer without affecting the TC sublayer. However due to the way cell delineation is accomplished, making such a change is not possible.

3.7. SUMMARY

The task of the data link layer is to convert the raw bit stream offered by the physical layer into a stream of frames for use by the network layer. Various framing methods are used, including character count, character stuffing, and bit stuffing. Data link protocols can provide error control to retransmit damaged or lost frames. To prevent a fast sender from overrunning a slow receiver, the data link protocol can also provide flow control. The sliding window mechanism is widely used to integrate error control and flow control in a convenient way.

Sliding window protocols can be categorized by the size of the sender's window and the size of the receiver's window. When both are equal to 1, the protocol is stop-and-wait. When the sender's window is greater than 1, for example to prevent the sender from blocking on a circuit with a long propagation delay, the receiver can be programmed either to discard all frames other than the next one in sequence (protocol 5) or buffer out of order frames until they are needed (protocol 6).

Protocols can be modeled using various techniques to help demonstrate their correctness (or lack thereof). Finite state machine models and Petri net models are commonly used for this purpose.

Many networks use one of the bit-oriented protocols—SDLC, HDLC, ADCCP, or LAPB—at the data link level. All of these protocols use flag bytes to delimit frames, and bit stuffing to prevent flag bytes from occurring in the data. All of them also use a sliding window for flow control. The Internet uses SLIP and PPP as data link protocols. ATM systems have their own simple protocol, which does a bare minimum of error checking and no flow control.

PROBLEMS

1. An upper layer message is split into 10 frames, each of which has an 80 percent chance of arriving undamaged. If no error control is done by the data link protocol, how many times must the message be sent on the average to get the entire thing through?

2. The following data fragment occurs in the middle of a data stream for which the character-stuffing algorithm described in the text is used: DLE, STX, A, DLE, B, DLE, ETX. What is the output after stuffing?

3. If the bit string 0111101111101111110 is bit stuffed, what is the output string?

4. When bit stuffing is used, is it possible for the loss, insertion, or modification of a single bit to cause an error not detected by the checksum? If not, why not? If so, how? Does the checksum length play a role here?

5. Can you think of any circumstances under which an open-loop protocol, (e.g., a Hamming code) might be preferable to the feedback type protocols discussed throughout this chapter?

6. To provide more reliability than a single parity bit can give, an error-detecting coding scheme uses one parity bit for checking all the odd numbered bits and a second parity bit for all the even numbered bits. What is the Hamming distance of this code?

7. One way of detecting errors is to transmit data as a block of n rows of k bits per row and adding parity bits to each row and each column. Will this scheme detect all single errors? Double errors? Triple errors?

8. A block of bits with n rows and k columns uses horizontal and vertical parity bits for error detection. Suppose that exactly 4 bits are inverted due to transmission errors. Derive an expression for the probability that the error will be undetected.

9. What is the remainder obtained by dividing $x^7 + x^5 + 1$ by the generator polynomial $x^3 + 1$?

10. Data link protocols almost always put the CRC in a trailer, rather than in a header. Why?

11. A channel has a bit rate of 4 kbps and a propagation delay of 20 msec. For what range of frame sizes does stop-and-wait give an efficiency of at least 50 percent?

12. A 3000-km long T1 trunk is used to transmit 64-byte frames using protocol 5. If the propagation speed is 6 μsec/km, how many bits should the sequence numbers be?

13. Imagine a sliding window protocol using so many bits for sequence numbers that wraparound never occurs. What relations must hold among the four window edges and the window size?

14. If the procedure *between* in protocol 5 checked for the condition $a \le b \le c$ instead of the condition $a \le b < c$, would that have any effect on the protocol's correctness or efficiency? Explain your answer.

15. In protocol 6, when a data frame arrives, a check is made to see if the sequence number differs from the one expected and *NoNak* is true. If both conditions hold, a NAK is sent. Otherwise, the auxiliary timer is started. Suppose that the else clause were omitted. Would this change affect the protocol's correctness?

16. Suppose that the three-statement while loop near the end of protocol 6 were removed from the code. Would this affect the correctness of the protocol or just the performance? Explain your answer.

17. Suppose that the case for checksum errors were removed from the switch statement of protocol 6. How would this change affect the operation of the protocol?

18. In protocol 6 the code for *FrameArrival* has a section used for NAKs. This section is invoked if the incoming frame is a NAK and another condition is met. Give a scenario where the presence of this other condition is essential.

19. Imagine that you are writing the data link layer software for a line used to send data to you, but not from you. The other end uses HDLC, with a 3-bit sequence number and a window size of seven frames. You would like to buffer as many out of sequence frames as possible to enhance efficiency, but you are not allowed to modify the software on the sending side. Is it possible to have a receiver window greater than one, and still guarantee that the protocol will never fail? If so, what is the largest window that can be safely used?

20. Consider the operation of protocol 6 over a 1-Mbps error-free line. The maximum frame size is 1000 bits. New packets are generated about 1 second apart. The timeout interval is 10 msec. If the special acknowledgement timer were eliminated, unnecessary timeouts would occur. How many times would the average message be transmitted?

21. In protocol 6 $MaxSeq = 2^n - 1$. While this condition is obviously desirable to make efficient use of header bits, we have not demonstrated that it is essential. Does the protocol work correctly for $MaxSeq = 4$, for example?

22. Frames of 1000 bits are sent over a 1-Mbps satellite channel. Acknowledgements are always piggybacked onto data frames. The headers are very short. Three-bit sequence numbers are used. What is the maximum achievable channel utilization for
(a) Stop-and-wait.
(b) Protocol 5.
(c) Protocol 6.

23. Compute the fraction of the bandwidth that is wasted on overhead (headers and retransmissions) for protocol 6 on a heavily loaded 50-kbps satellite channel with data frames consisting of 40 header and 3960 data bits. ACK frames never occur. NAK frames are 40 bits. The error rate for data frames is 1 percent, and the error rate for NAK frames is negligible. The sequence numbers are 8 bits.

24. Consider an error-free 64-kbps satellite channel used to send 512-byte data frames in one direction, with very short acknowledgements coming back the other way. What is the maximum throughput for window sizes of 1, 7, 15, and 127?

25. A 100 km long cable runs at the T1 data rate. The propagation speed in the cable is 2/3 the speed of light. How many bits fit in the cable?

26. Redraw Fig. 3-21 for a full-duplex channel that never loses frames. Is the protocol failure still possible?

27. Give the firing sequence for the Petri net of Fig. 3-23 corresponding to the state sequence (000), (01A), (01—), (010), (01A) in Fig. 3-20. Explain in words what the sequence represents.

28. Given the transition rules $AC \rightarrow B$, $B \rightarrow AC$, $CD \rightarrow E$, and $E \rightarrow CD$, draw the Petri net described. From the Petri net, draw the finite state graph reachable from the initial state ACD. What well-known computer science concept do these transition rules model?

29. PPP is based closely on HDLC, which uses bit stuffing to prevent accidental flag bytes within the payload from causing confusion. Give at least one reason why PPP uses character stuffing instead.

30. What is the minimum overhead in sending an IP packet using PPP? Count only the overhead introduced by PPP itself, not the IP header overhead.

31. Consider the ATM cell delineation heuristic with $\alpha = 5$, $\delta = 6$, and a per-bit error rate of 10^{-5}. Once the system is synchronized, how long will it remain so, despite occasional header bit errors? Assume the line is running at OC-3.

32. Write a program to stochastically simulate the behavior of a Petri net. The program should read in the transition rules as well as a list of states corresponding to the network link layer issuing a new packet or the accepting a new packet. From the initial state, also read in, the program should pick enabled transitions at random and fire them, checking to see if a host ever accepts two messages without the other host emitting a new one in between.

4

THE MEDIUM ACCESS SUBLAYER

As we pointed out in Chap. 1, networks can be divided into two categories: those using point-to-point connections and those using broadcast channels. This chapter deals with broadcast networks and their protocols.

In any broadcast network, the key issue is how to determine who gets to use the channel when there is competition for it. To make this point clearer, consider a conference call in which six people, on six different telephones, are all connected together so that each one can hear and talk to all the others. It is very likely that when one of them stops speaking, two or more will start talking at once, leading to chaos. In a face-to-face meeting, chaos is avoided by external means, for example, at a meeting, people raise their hands to request permission to speak. When only a single channel is available, determining who should go next is much harder. Many protocols for solving the problem are known and form the contents of this chapter. In the literature, broadcast channels are sometimes referred to as **multiaccess channels** or **random access channels**.

The protocols used to determine who goes next on a multiaccess channel belong to a sublayer of the data link layer called the **MAC** (**Medium Access Control**) sublayer. The MAC sublayer is especially important in LANs, nearly all of which use a multiaccess channel as the basis of their communication. WANs, in contrast, use point-to-point links, except for satellite networks. Because multiaccess channels and LANs are so closely related, in this chapter we will discuss LANs in general, as well as satellite and some other broadcast networks.

Technically, the MAC sublayer is the bottom part of the data link layer, so logically we should have studied it before examining all the point-to-point protocols in Chap. 3. Nevertheless, for most people, understanding protocols involving multiple parties is easier after two-party protocols are well understood. For that reason we have deviated slightly from a strict bottom-up order of presentation.

4.1. THE CHANNEL ALLOCATION PROBLEM

The central theme of this chapter is how to allocate a single broadcast channel among competing users. We will first look at static and dynamic schemes in general. Then we will examine a number of specific algorithms.

4.1.1. Static Channel Allocation in LANs and MANs

The traditional way of allocating a single channel, such as a telephone trunk, among multiple competing users is Frequency Division Multiplexing (FDM). If there are N users, the bandwidth is divided into N equal sized portions (see Fig. 2-24), each user being assigned one portion. Since each user has a private frequency band, there is no interference between users. When there is only a small and fixed number of users, each of which has a heavy (buffered) load of traffic (e.g., carriers' switching offices), FDM is a simple and efficient allocation mechanism.

However, when the number of senders is large and continuously varying, or the traffic is bursty, FDM presents some problems. If the spectrum is cut up into N regions, and fewer than N users are currently interested in communicating, a large piece of valuable spectrum will be wasted. If more than N users want to communicate, some of them will be denied permission, for lack of bandwidth, even if some of the users who have been assigned a frequency band hardly ever transmit or receive anything.

However, even assuming that the number of users could somehow be held constant at N, dividing the single available channel into static subchannels is inherently inefficient. The basic problem is that when some users are quiescent, their bandwidth is simply lost. They are not using it, and no one else is allowed to use it either. Furthermore, in most computer systems, data traffic is extremely bursty (peak traffic to mean traffic ratios of 1000:1 are common). Consequently, most of the channels will be idle most of the time.

The poor performance of static FDM can easily be seen from a simple queueing theory calculation. Let us start with the mean time delay, T, for a channel of capacity C bps, with an arrival rate of λ frames/sec, each frame having a length drawn from an exponential probability density function with mean $1/\mu$ bits/frame:

$$T = \frac{1}{\mu C - \lambda}$$

Now let us divide the single channel up into N independent subchannels, each with capacity C/N bps. The mean input rate on each of the subchannels will now be λ/N. Recomputing T we get

$$T_{\mathrm{FDM}} = \frac{1}{\mu(C/N) - (\lambda/N)} = \frac{N}{\mu C - \lambda} = NT \qquad (4\text{-}1)$$

The mean delay using FDM is N times worse than if all the frames were somehow magically arranged orderly in a big central queue.

Precisely the same arguments that apply to FDM also apply to time division multiplexing (TDM). Each user is statically allocated every Nth time slot. If a user does not use the allocated slot, it just lies fallow. Since none of the traditional static channel allocation methods work well with bursty traffic, we will now explore dynamic methods.

4.1.2. Dynamic Channel Allocation in LANs and MANs

Before we get into the first of the many channel allocation methods to be discussed in this chapter, it is worthwhile carefully formulating the allocation problem. Underlying all the work done in this area are five key assumptions, described below.

1. **Station model**. The model consists of N independent **stations** (computers, telephones, personal communicators, etc.), each with a program or user that generates frames for transmission. The probability of a frame being generated in an interval of length Δt is $\lambda \Delta t$, where λ is a constant (the arrival rate of new frames). Once a frame has been generated, the station is blocked and does nothing until the frame has been successfully transmitted.

2. **Single Channel Assumption**. A single channel is available for all communication. All stations can transmit on it and all can receive from it. As far as the hardware is concerned, all stations are equivalent, although protocol software may assign priorities to them.

3. **Collision Assumption**. If two frames are transmitted simultaneously, they overlap in time and the resulting signal is garbled. This event is called a **collision**. All stations can detect collisions. A collided frame must be transmitted again later. There are no errors other than those generated by collisions.

4a. **Continuous Time**. Frame transmission can begin at any instant. There is no master clock dividing time into discrete intervals.

4b. **Slotted Time**. Time is divided into discrete intervals (slots). Frame transmissions always begin at the start of a slot. A slot may contain 0, 1, or more frames, corresponding to an idle slot, a successful transmission, or a collision, respectively.

5a. **Carrier Sense**. Stations can tell if the channel is in use before trying to use it. If the channel is sensed as busy, no station will attempt to use it until it goes idle.

5b. **No Carrier Sense**. Stations cannot sense the channel before trying to use it. They just go ahead and transmit. Only later can they determine whether or not the transmission was successful.

Some discussion of these assumptions is in order. The first one says that stations are independent, and that work is generated at a constant rate. It also implicitly assumes that each station only has one program or user, so while the station is blocked, no new work is generated. More sophisticated models allow multiprogrammed stations that can generate work while a station is blocked, but the analysis of these stations is much more complex.

The single channel assumption is the heart of the matter. There are no external ways to communicate. Stations cannot raise their hands to request that the teacher call on them.

The collision assumption is also basic, although in some systems (notably spread spectrum), this assumption is relaxed, with surprising results. Also, some LANs, such as token rings, use a mechanism for contention elimination that eliminates collisions.

There are two alternative assumptions about time. Either it is continuous or it is slotted. Some systems use one and some systems use the other, so we will discuss and analyze both. Obviously, for a given system, only one of them holds.

Similarly, a network can either have carrier sensing or not have it. LANs generally have carrier sense, but satellite networks do not (due to the long propagation delay). Stations on carrier sense networks can terminate their transmission prematurely if they discover that it is colliding with another transmission. Note that the word "carrier" in this sense refers to an electrical signal on the cable and has nothing to do with the common carriers (e.g., telephone companies) that date back to the Pony Express days.

4.2. MULTIPLE ACCESS PROTOCOLS

Many algorithms for allocating a multiple access channel are known. In the following sections we will study a representative sample of the more interesting ones and give some examples of their use.

4.2.1. ALOHA

In the 1970s, Norman Abramson and his colleagues at the University of Hawaii devised a new and elegant method to solve the channel allocation problem. Their work has been extended by many researchers since then (Abramson, 1985). Although Abramson's work, called the ALOHA system, used ground-based radio broadcasting, the basic idea is applicable to any system in which uncoordinated users are competing for the use of a single shared channel.

We will discuss two versions of ALOHA here: pure and slotted. They differ with respect to whether or not time is divided up into discrete slots into which all frames must fit. Pure ALOHA does not require global time synchronization; slotted ALOHA does.

Pure ALOHA

The basic idea of an ALOHA system is simple: let users transmit whenever they have data to be sent. There will be collisions, of course, and the colliding frames will be destroyed. However, due to the feedback property of broadcasting, a sender can always find out whether or not its frame was destroyed by listening to the channel, the same way other users do. With a LAN, the feedback is immediate; with a satellite, there is a delay of 270 msec before the sender knows if the transmission was successful. If the frame was destroyed, the sender just waits a random amount of time and sends it again. The waiting time must be random or the same frames will collide over and over, in lockstep. Systems in which multiple users share a common channel in a way that can lead to conflicts are widely known as **contention** systems.

A sketch of frame generation in an ALOHA system is given in Fig. 4-1. We have made the frames all the same length because the throughput of ALOHA systems is maximized by having a uniform frame size rather than allowing variable length frames.

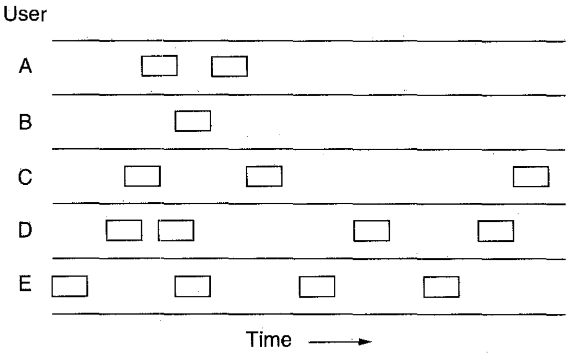


Fig. 4-1. In pure ALOHA, frames are transmitted at completely arbitrary times.

Whenever two frames try to occupy the channel at the same time, there will be a collision and both will be garbled. If the first bit of a new frame overlaps with just the last bit of a frame almost finished, both frames will be totally destroyed, and both will have to be retransmitted later. The checksum cannot (and should not) distinguish between a total loss and a near miss. Bad is bad.

A most interesting question is: What is the efficiency of an ALOHA channel? That is, what fraction of all transmitted frames escape collisions under these chaotic circumstances? Let us first consider an infinite collection of interactive users sitting at their computers (stations). A user is always in one of two states: typing or waiting. Initially, all users are in the typing state. When a line is finished, the user stops typing, waiting for a response. The station then transmits a frame containing the line and checks the channel to see if it was successful. If so, the user sees the reply and goes back to typing. If not, the user continues to wait and the frame is retransmitted over and over until it has been successfully sent.

Let the "frame time" denote the amount of time needed to transmit the standard, fixed-length frame (i.e., the frame length divided by the bit rate). At this point we assume that the infinite population of users generates new frames according to a Poisson distribution with mean S frames per frame time. (The infinite-population assumption is needed to ensure that S does not decrease as users become blocked.) If $S > 1$, the user community is generating frames at a higher rate than the channel can handle, and nearly every frame will suffer a collision. For reasonable throughput we would expect $0 < S < 1$.

In addition to the new frames, the stations also generate retransmissions of frames that previously suffered collisions. Let us further assume that the probability of k transmission attempts per frame time, old and new combined, is also Poisson, with mean G per frame time. Clearly, $G \geq S$. At low load (i.e., $S \approx 0$), there will be few collisions, hence few retransmissions, so $G \approx S$. At high load there will be many collisions, so $G > S$. Under all loads, the throughput is just the offered load, G, times the probability of a transmission being successful—that is, $S = GP_0$, where P_0 is the probability that a frame does not suffer a collision.

A frame will not suffer a collision if no other frames are sent within one frame time of its start, as shown in Fig. 4-2. Under what conditions will the shaded frame arrive undamaged? Let t be the time required to send a frame. If any other user has generated a frame between time t_0 and $t_0 + t$, the end of that frame will collide with the beginning of the shaded one. In fact, the shaded frame's fate was already sealed even before the first bit was sent, but since in pure ALOHA a station does not listen to the channel before transmitting, it has no way of knowing that another frame was already underway. Similarly, any other frame started between $t_0 + t$ and $t_0 + 2t$ will bump into the end of the shaded frame.

The probability that k frames are generated during a given frame time is given by the Poisson distribution:

$$\Pr[k] = \frac{G^k e^{-G}}{k!} \tag{4-2}$$

so the probability of zero frames is just e^{-G}. In an interval two frame times long, the mean number of frames generated is $2G$. The probability of no other traffic

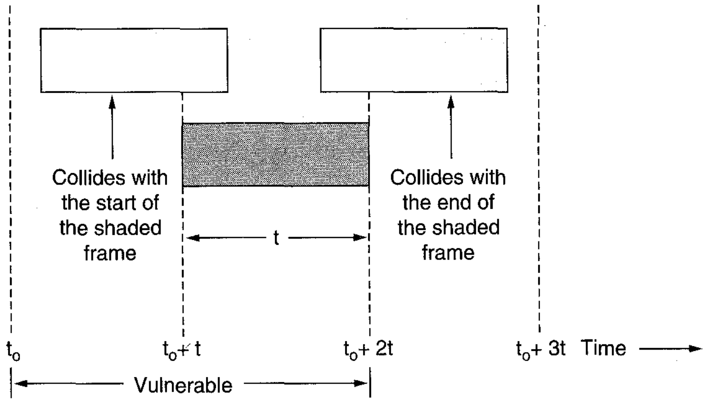

Collides with the start of the shaded frame
Collides with the end of the shaded frame
t
t_0
t_0+ t
t_0+ 2t
t_0+ 3t
Time
Vulnerable

Fig. 4-2. Vulnerable period for the shaded frame.

being initiated during the entire vulnerable period is thus given by $P_0 = e^{-2G}$. Using $S = GP_0$, we get

$$S = Ge^{-2G}$$

The relation between the offered traffic and the throughput is shown in Fig. 4-3. The maximum throughput occurs at $G = 0.5$, with $S = 1/2e$, which is about 0.184. In other words, the best we can hope for is a channel utilization of 18 percent. This result is not very encouraging, but with everyone transmitting at will, we could hardly have expected a 100 percent success rate.

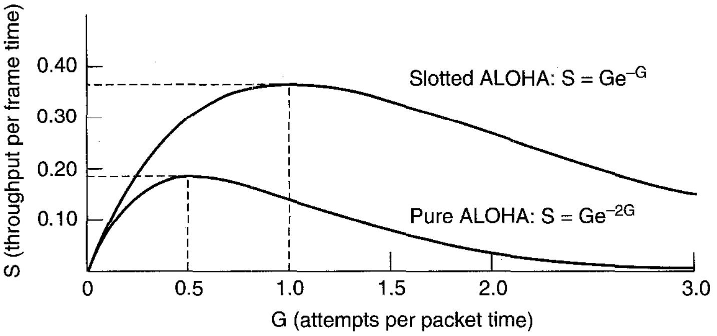


Fig. 4-3. Throughput versus offered traffic for ALOHA systems.

Slotted ALOHA

In 1972, Roberts published a method for doubling the capacity of an ALOHA system (Roberts, 1972). His proposal was to divide time up into discrete intervals, each interval corresponding to one frame. This approach requires the users to agree of slot boundaries. One way to achieve synchronization would be to have one special station emit a pip at the start of each interval, like a clock.

In Roberts' method, which has come to be known as **slotted ALOHA**, in contrast to Abramson's **pure ALOHA**, a computer is not permitted to send whenever a carriage return is typed. Instead, it is required to wait for the beginning of the next slot. Thus the continuous pure ALOHA is turned into a discrete one. Since the vulnerable period is now halved, the probability of no other traffic during the same slot as our test frame is e^{-G} which leads to

$$S = Ge^{-G} \tag{4-3}$$

As you can see from Fig. 4-3, slotted ALOHA peaks at $G = 1$, with a throughput of $S = 1/e$ or about 0.368, twice that of pure ALOHA. If the system is operating at $G = 1$, the probability of an empty slot is 0.368 (from Eq. 4-2). The best we can hope for using slotted ALOHA is 37 percent of the slots empty, 37 percent successes, and 26 percent collisions. Operating at higher values of G reduces the number of empties but increases the number of collisions exponentially. To see how this rapid growth of collisions with G comes about, consider the transmission of a test frame. The probability that it will avoid a collision is e^{-G}, the probability that all the other users are silent in that slot. The probability of a collision is then just $1 - e^{-G}$. The probability of a transmission requiring exactly k attempts, (i.e., $k - 1$ collisions followed by one success) is

$$P_k = e^{-G}(1 - e^{-G})^{k-1}$$

The expected number of transmissions, E, per carriage return typed is then

$$E = \sum_{k=1}^{\infty} kP_k = \sum_{k=1}^{\infty} ke^{-G}(1 - e^{-G})^{k-1} = e^G$$

As a result of the exponential dependence of E upon G, small increases in the channel load can drastically reduce its performance.

4.2.2. Carrier Sense Multiple Access Protocols

With slotted ALOHA the best channel utilization that can be achieved is $1/e$. This is hardly surprising, since with stations transmitting at will, without paying attention to what the other stations are doing, there are bound to be many collisions. In local area networks, however it is possible for stations to detect what other stations are doing, and adapt their behavior accordingly. These networks can achieve a much better utilization than $1/e$. In this section we will discuss some protocols for improving performance.

Protocols in which stations listen for a carrier (i.e., a transmission) and act accordingly are called **carrier sense protocols**. A number of them have been proposed. Kleinrock and Tobagi (1975) have analyzed several such protocols in detail. Below we will mention several versions of the carrier sense protocols.

Persistent and Nonpersistent CSMA

The first carrier sense protocol that we will study here is called **1-persistent CSMA** (Carrier Sense Multiple Access). When a station has data to send, it first listens to the channel to see if anyone else is transmitting at that moment. If the channel is busy, the station waits until it becomes idle. When the station detects an idle channel, it transmits a frame. If a collision occurs, the station waits a random amount of time and starts all over again. The protocol is called 1-persistent because the station transmits with a probability of 1 whenever it finds the channel idle.

The propagation delay has an important effect on the performance of the protocol. There is a small chance that just after a station begins sending, another station will become ready to send and sense the channel. If the first station's signal has not yet reached the second one, the latter will sense an idle channel and will also begin sending, resulting in a collision. The longer the propagation delay, the more important this effect becomes, and the worse the performance of the protocol.

Even if the propagation delay is zero, there will still be collisions. If two stations become ready in the middle of a third station's transmission, both will wait politely until the transmission ends and then both will begin transmitting exactly simultaneously, resulting in a collision. If they were not so impatient, there would be fewer collisions. Even so, this protocol is far better than pure ALOHA, because both stations have the decency to desist from interfering with the third station's frame. Intuitively, this will lead to a higher performance than pure ALOHA. Exactly the same holds for slotted ALOHA.

A second carrier sense protocol is **nonpersistent CSMA**. In this protocol, a conscious attempt is made to be less greedy than in the previous one. Before sending, a station senses the channel. If no one else is sending, the station begins doing so itself. However, if the channel is already in use, the station does not continually sense it for the purpose of seizing it immediately upon detecting the end of the previous transmission. Instead, it waits a random period of time and then repeats the algorithm. Intuitively this algorithm should lead to better channel utilization and longer delays than 1-persistent CSMA.

The last protocol is **p-persistent CSMA**. It applies to slotted channels and works as follows. When a station becomes ready to send, it senses the channel. If it is idle, it transmits with a probability p. With a probability $q = 1 - p$ it defers until the next slot. If that slot is also idle, it either transmits or defers again, with probabilities p and q. This process is repeated until either the frame has been transmitted or another station has begun transmitting. In the latter case, it acts as if there had been a collision (i.e., it waits a random time and starts again). If the station initially senses the channel busy, it waits until the next slot and applies the above algorithm. Figure 4-4 shows the throughput versus offered traffic for all three protocols, as well as pure and slotted ALOHA.

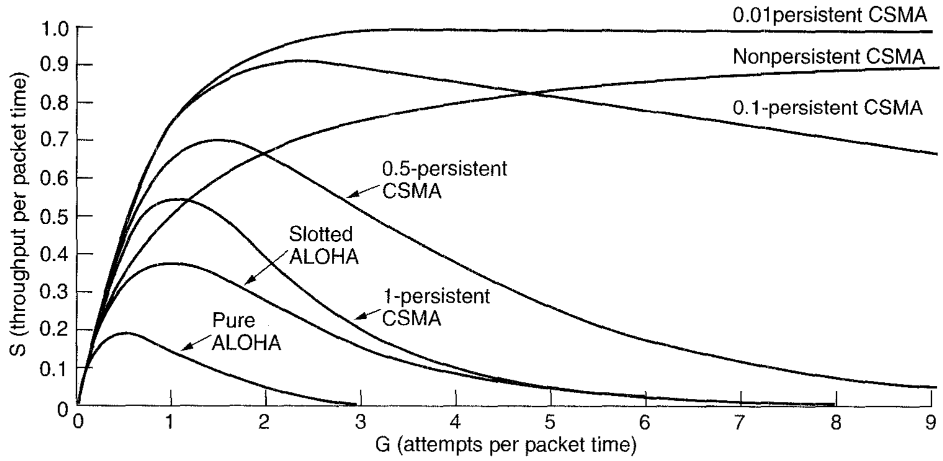


Fig. 4-4. Comparison of the channel utilization versus load for various random access protocols.

CSMA with Collision Detection

Persistent and nonpersistent CSMA protocols are clearly an improvement over ALOHA because they ensure that no station begins to transmit when it senses the channel busy. Another improvement is for stations to abort their transmissions as soon as they detect a collision. In other words, if two stations sense the channel to be idle and begin transmitting simultaneously, they will both detect the collision almost immediately. Rather than finish transmitting their frames, which are irretrievably garbled anyway, they should abruptly stop transmitting as soon as the collision is detected. Quickly terminating damaged frames saves time and bandwidth. This protocol, known as **CSMA/CD** (**Carrier Sense Multiple Access with Collision Detection**), is widely used on LANs in the MAC sublayer.

CSMA/CD, as well as many other LAN protocols, uses the conceptual model of Fig. 4-5. At the point marked t_0, a station has finished transmitting its frame. Any other station having a frame to send may now attempt to do so. If two or more stations decide to transmit simultaneously, there will be a collision. Collisions can be detected by looking at the power or pulse width of the received signal and comparing it to the transmitted signal.

After a station detects a collision, it aborts its transmission, waits a random period of time, and then tries again, assuming that no other station has started transmitting in the meantime. Therefore, our model for CSMA/CD will consist of alternating contention and transmission periods, with idle periods occurring when all stations are quiet (e.g., for lack of work).

Now let us look closely at the details of the contention algorithm. Suppose

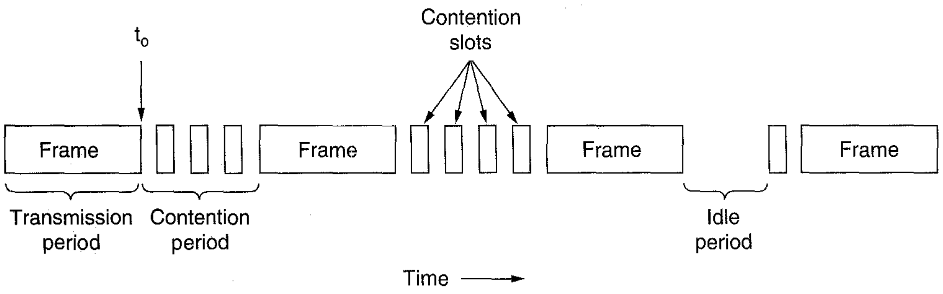


Fig. 4-5. CSMA/CD can be in one of three states: contention, transmission, or idle.

that two stations both begin transmitting at exactly time t_0. How long will it take them to realize that there has been a collision? The answer to this question is vital to determining the length of the contention period, and hence what the delay and throughput will be. The minimum time to detect the collision is then just the time it takes the signal to propagate from one station to the other.

Based on this reasoning, you might think that a station not hearing a collision for a time equal to the full cable propagation time after starting its transmission could be sure it had seized the cable. By "seized," we mean that all other stations knew it was transmitting and would not interfere. This conclusion is wrong. Consider the following worst-case scenario. Let the time for a signal to propagate between the two farthest stations be τ. At t_0, one station begins transmitting. At $\tau - \varepsilon$, an instant before the signal arrives at the most distant station, that station also begins transmitting. Of course, it detects the collision almost instantly and stops, but the little noise burst caused by the collision does not get back to the original station until time $2\tau - \varepsilon$. In other words, in the worst case a station cannot be sure that it has seized the channel until it has transmitted for 2τ without hearing a collision. For this reason we will model the contention interval as a slotted ALOHA system with slot width 2τ. On a 1-km long coaxial cable, $\tau \approx 5$ μsec. For simplicity we will assume that each slot contains just 1 bit. Once the channel has been seized, a station can transmit at any rate it wants to, of course, not just at 1 bit per 2τ sec.

It is important to realize that collision detection is an *analog* process. The station's hardware must listen to the cable while it is transmitting. If what it reads back is different from what it is putting out, it knows a collision is occurring. The implication is that the signal encoding must allow collisions to be detected (e.g., a collision of two 0-volt signals may well be impossible to detect). For this reason, special encoding is commonly used.

CSMA/CD is an important protocol. Later in this chapter we will study one version of it, IEEE 802.3 (Ethernet), which is an international standard.

To avoid any misunderstanding, it is worth noting that no MAC-sublayer protocol guarantees reliable delivery. Even in the absence of collisions, the receiver may not have copied the frame correctly due to various reasons (e.g., lack of buffer space or a missed interrupt).

4.2.3. Collision-Free Protocols

Although collisions do not occur with CSMA/CD once a station has unambiguously seized the channel, they can still occur during the contention period. These collisions adversely affect the system performance, especially when the cable is long (i.e., large τ) and the frames short. As very long, high-bandwidth fiber optic networks come into use, the combination of large τ and short frames will become an increasingly serious problem. In this section, we will examine some protocols that resolve the contention for the channel without any collisions at all, not even during the contention period.

In the protocols to be described, we make the assumption that there are N stations, each with a unique address from 0 to $N - 1$ "wired" into it. That some stations may be inactive part of the time does not matter. The basic question remains: Which station gets the channel after a successful transmission? We continue using the model of Fig. 4-5 with its discrete contention slots.

A Bit-Map Protocol

In our first collision-free protocol, the **basic bit-map method**, each contention period consists of exactly N slots. If station 0 has a frame to send, it transmits a 1 bit during the zeroth slot. No other station is allowed to transmit during this slot. Regardless of what station 0 does, station 1 gets the opportunity to transmit a 1 during slot 1, but only if it has a frame queued. In general, station j may announce the fact that it has a frame to send by inserting a 1 bit into slot j. After all N slots have passed by, each station has complete knowledge of which stations wish to transmit. At that point, they begin transmitting in numerical order (see Fig. 4-6).

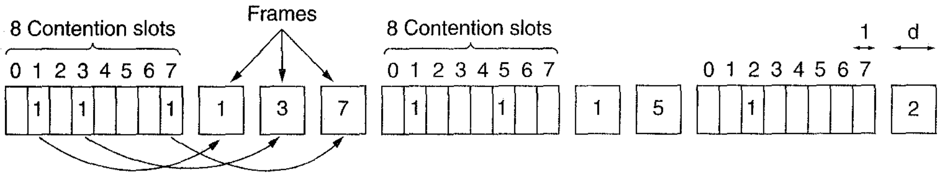


Fig. 4-6. The basic bit-map protocol.

Since everyone agrees on who goes next, there will never be any collisions. After the last ready station has transmitted its frame, an event all stations can easily monitor, another N bit contention period is begun. If a station becomes ready just after its bit slot has passed by, it is out of luck and must remain silent until every station has had a chance and the bit map has come around again. Protocols like this in which the desire to transmit is broadcast before the actual transmission are called **reservation protocols**.

Let us briefly analyze the performance of this protocol. For convenience, we will measure time in units of the contention bit slot, with data frames consisting of d time units. Under conditions of low load, the bit map will simply be repeated over and over, for lack of data frames.

Consider the situation from the point of view of a low-numbered station, such as 0 or 1. Typically, when it becomes ready to send, the "current" slot will be somewhere in the middle of the bit map. On the average, the station will have to wait $N/2$ slots for the current scan to finish and another full N slots for the following scan to run to completion before it may begin transmitting.

The prospects for high-numbered stations are brighter. Generally, these will only have to wait half a scan ($N/2$ bit slots) before starting to transmit. High-numbered stations rarely have to wait for the next scan. Since low-numbered stations must wait on the average $1.5N$ slots and high-numbered stations must wait on the average $0.5N$ slots, the mean for all stations is N slots. The channel efficiency at low load is easy to compute. The overhead per frame is N bits, and the amount of data is d bits, for an efficiency of $d/(N + d)$.

At high load, when all the stations have something to send all the time, the N bit contention period is prorated over N frames, yielding an overhead of only 1 bit per frame, or an efficiency of $d/(d + 1)$. The mean delay for a frame is equal to the sum of the time it queues inside its station, plus an additional $N(d + 1)/2$ once it gets to the head of its internal queue.

Binary Countdown

A problem with the basic bit-map protocol is that the overhead is 1 bit per station. We can do better than that by using binary station addresses. A station wanting to use the channel now broadcasts its address as a binary bit string, starting with the high-order bit. All addresses are assumed to be the same length. The bits in each address position from different stations are BOOLEAN ORed together. We will call this protocol **binary countdown**. It is used in Datakit (Fraser, 1987).

To avoid conflicts, an arbitration rule must be applied: as soon as a station sees that a high-order bit position that is 0 in its address has been overwritten with a 1, it gives up. For example, if stations 0010, 0100, 1001, and 1010 are all trying to get the channel, in the first bit time the stations transmit 0, 0, 1, and 1, respectively. These are ORed together to form a 1. Stations 0010 and 0100 see the 1 and know that a higher-numbered station is competing for the channel, so they give up for the current round. Stations 1001 and 1010 continue.

The next bit is 0, and both stations continue. The next bit is 1, so station 1001 gives up. The winner is station 1010, because it has the highest address. After winning the bidding, it may now transmit a frame, after which another bidding cycle starts. The protocol is illustrated in Fig. 4-7.

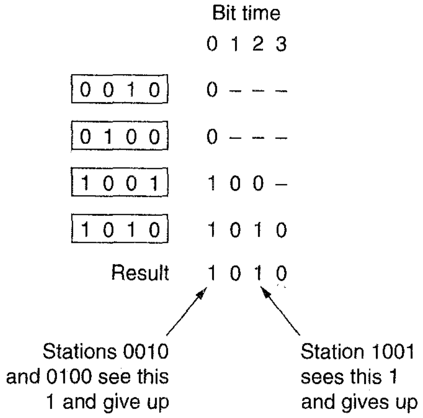


Fig. 4-7. The binary countdown protocol. A dash indicates silence.

The channel efficiency of this method is $d/(d + \ln N)$. If, however, the frame format has been cleverly chosen so that the sender's address is the first field in the frame, even these $\ln N$ bits are not wasted, and the efficiency is 100 percent.

Mok and Ward (1979) have described a variation of binary countdown using a parallel rather than a serial interface. They also suggest using virtual station numbers, with the virtual station numbers from 0 up to and including the successful station being circularly permuted after each transmission, in order to give higher priority to stations that have been silent unusually long. For example, if stations *C*, *H*, *D*, *A*, *G*, *B*, *E*, *F* have priorities 7, 6, 5, 4, 3, 2, 1, and 0, respectively, then a successful transmission by *D* puts it at the end of the list, giving a priority order of *C*, *H*, *A*, *G*, *B*, *E*, *F*, *D*. Thus *C* remains virtual station 7, but *A* moves up from 4 to 5 and *D* drops from 5 to 0. Station *D* will now only be able to acquire the channel if no other station wants it.

4.2.4. Limited-Contention Protocols

We have now considered two basic strategies for channel acquisition in a cable network: contention, as in CSMA, and collision-free methods. Each strategy can be rated as to how well it does with respect to the two important performance measures, delay at low load and channel efficiency at high load. Under conditions of light load, contention (i.e., pure or slotted ALOHA) is preferable due to its low delay. As the load increases, contention becomes increasingly less attractive, because the overhead associated with channel arbitration becomes greater. Just the reverse is true for the collision-free protocols. At low load, they have high delay, but as the load increases, the channel efficiency improves rather than gets worse as it does for contention protocols.

Obviously, it would be nice if we could combine the best properties of the contention and collision-free protocols, arriving at a new protocol that used contention at low loads to provide low delay, but used a collision-free technique at high load to provide good channel efficiency. Such protocols, which we will call **limited contention protocols**, do, in fact, exist, and will conclude our study of carrier sense networks.

Up until now the only contention protocols we have studied have been symmetric, that is, each station attempts to acquire the channel with some probability, p, with all stations using the same p. Interestingly enough, the overall system performance can sometimes be improved by using a protocol that assigns different probabilities to different stations.

Before looking at the asymmetric protocols, let us quickly review the performance of the symmetric case. Suppose that k stations are contending for channel access. Each has a probability p of transmitting during each slot. The probability that some station successfully acquires the channel during a given slot is then $kp(1-p)^{k-1}$. To find the optimal value of p, we differentiate with respect to p, set the result to zero, and solve for p. Doing so, we find that the best value of p is $1/k$. Substituting $p = 1/k$ we get

$$\Pr[\text{success with optimal } p] = \left(\frac{k-1}{k}\right)^{k-1} \qquad (4\text{-}4)$$

This probability is plotted in Fig. 4-8. For small numbers of stations, the chances of success are good, but as soon as the number of stations reaches even five, the probability has dropped close to its asymptotic value of $1/e$.

From Fig. 4-8, it is fairly obvious that the probability of some station acquiring the channel can be increased only by decreasing the amount of competition. The limited-contention protocols do precisely that. They first divide the stations up into (not necessarily disjoint) groups. Only the members of group 0 are permitted to compete for slot 0. If one of them succeeds, it acquires the channel and transmits its frame. If the slot lies fallow or if there is a collision, the members of group 1 contend for slot 1, etc. By making an appropriate division of stations into groups, the amount of contention for each slot can be reduced, thus operating each slot near the left end of Fig. 4-8.

The trick is how to assign stations to slots. Before looking at the general case, let us consider some special cases. At one extreme, each group has but one member. Such an assignment guarantees that there will never be collisions, because at most one station is contending per slot. We have seen such protocols before (e.g., binary countdown). The next special case is to assign two stations per group. The probability that both will try to transmit during a slot is p^2, which

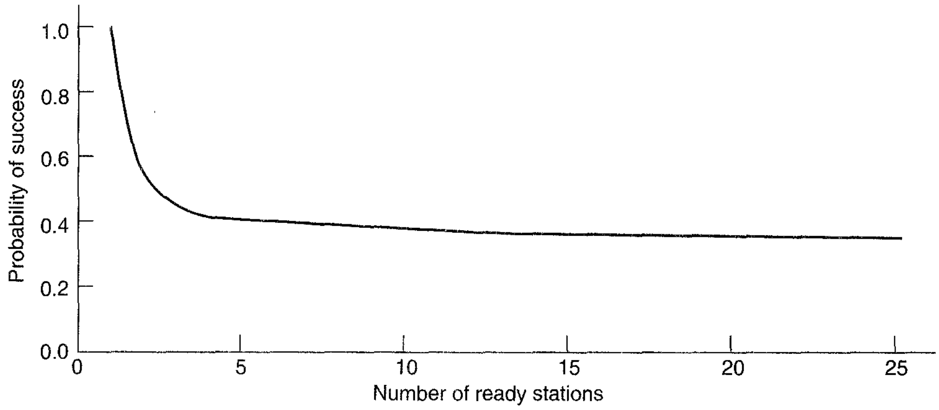


Fig. 4-8. Acquisition probability for a symmetric contention channel.

for small p is negligible. As more and more stations are assigned to the same slot, the probability of a collision grows, but the length of the bit-map scan needed to give everyone a chance shrinks. The limiting case is a single group containing all stations (i.e., slotted ALOHA). What we need is a way to assign stations to slots dynamically, with many stations per slot when the load is low and few (or even just one) station per slot when the load is high.

The Adaptive Tree Walk Protocol

One particularly simple way of performing the necessary assignment is to use the algorithm devised by the U.S. Army for testing soldiers for syphilis during World War II (Dorfman, 1943). In short, the Army took a blood sample from N soldiers. A portion of each sample was poured into a single test tube. This mixed sample was then tested for antibodies. If none were found, all the soldiers in the group were declared healthy. If antibodies were present, two new mixed samples were prepared, one from soldiers 1 through $N/2$ and one from the rest. The process was repeated recursively until the infected soldiers were determined.

For the computer version of this algorithm (Capetanakis, 1979) it is convenient to think of the stations as the leaves of a binary tree, as illustrated in Fig. 4-9. In the first contention slot following a successful frame transmission, slot 0, all stations are permitted to try to acquire the channel. If one of them does so, fine. If there is a collision, then during slot 1 only those stations falling under node 2 in the tree may compete. If one of them acquires the channel, the slot following the frame is reserved for those stations under node 3. If, on the other hand, two or more stations under node 2 want to transmit, there will be a collision during slot 1, in which case it is node 4's turn during slot 2.

In essence, if a collision occurs during slot 0, the entire tree is searched, depth

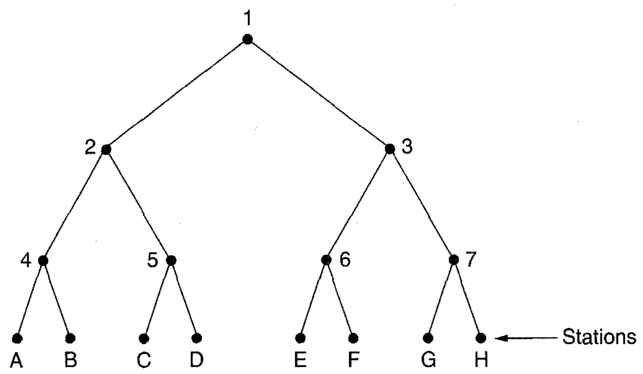


Fig. 4-9. The tree for eight stations.

first, to locate all ready stations. Each bit slot is associated with some particular node in the tree. If a collision occurs, the search continues recursively with the node's left and right children. If a bit slot is idle or if there is only one station that transmits in it, the searching of its node can stop, because all ready stations have been located. (Were there more than one, there would have been a collision.)

When the load on the system is heavy, it is hardly worth the effort to dedicate slot 0 to node 1, because that makes sense only in the unlikely event that precisely one station has a frame to send. Similarly, one could argue that nodes 2 and 3 should be skipped as well for the same reason. Put in more general terms, at what level in the tree should the search begin? Clearly, the heavier the load, the farther down the tree the search should begin. We will assume that each station has a good estimate of the number of ready stations, q, for example, from monitoring recent traffic.

To proceed, let us number the levels of the tree from the top, with node 1 in Fig. 4-9 at level 0, nodes 2 and 3 at level 1, etc. Notice that each node at level i has a fraction 2^{-i} of the stations below it. If the q ready stations are uniformly distributed, the expected number of them below a specific node at level i is just $2^{-i}q$. Intuitively, we would expect the optimal level to begin searching the tree as the one at which the mean number of contending stations per slot is 1, that is, the level at which $2^{-i}q = 1$. Solving this equation we find that $i = \log_2 q$.

Numerous improvements to the basic algorithm have been discovered and are discussed in some detail by Bertsekas and Gallager (1992). For example, consider the case of stations G and H being the only ones wanting to transmit. At node 1 a collision will occur, so 2 will be tried and discovered idle. It is pointless to probe node 3 since it is guaranteed to have a collision (we know that two or more stations under 1 are ready and none of them are under 2 so they must all be under 3). The probe of 3 can be skipped and 6 tried next. When this probe also turns up nothing, 7 can be skipped and node G tried next.

4.2.5. Wavelength Division Multiple Access Protocols

A different approach to channel allocation is to divide the channel into subchannels using FDM, TDM, or both, and dynamically allocate them as needed. Schemes like this are commonly used on fiber optic LANs in order to permit different conversations to use different wavelengths (i.e., frequencies) at the same time. In this section we will examine one such protocol (Humblet et al., 1992).

A simple way to build an all-optical LAN is to use a passive star coupler (see Fig. 2-10). In effect, two fibers from each station are fused to a glass cylinder. One fiber is for output to the cylinder and one is for input from the cylinder. Light output by any station illuminates the cylinder and can be detected by all the other stations. Passive stars can handle hundreds of stations.

To allow multiple transmissions at the same time, the spectrum is divided up into channels (wavelength bands), as shown in Fig. 2-24. In this protocol, **WDMA (Wavelength Division Multiple Access)**, each station is assigned two channels. A narrow channel is provided as a control channel to signal the station, and a wide channel is provided so the station can output data frames.

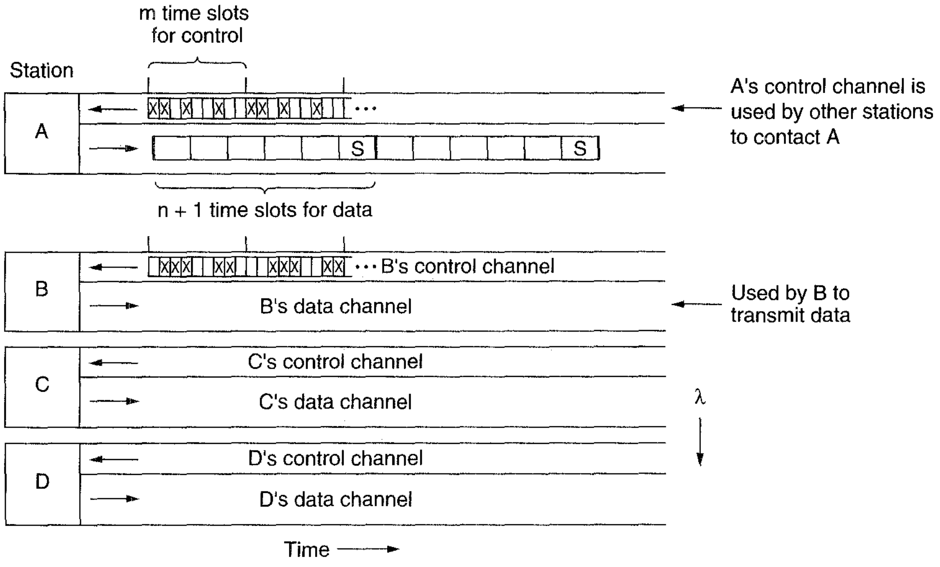


Fig. 4-10. Wavelength division multiple access.

Each channel is divided up into groups of time slots, as depicted in Fig. 4-10. Let us call the number of slots in the control channel m and the number of slots in the data channel $n + 1$, where n of these are for data and the last one is used by the station to report on its status (mainly, which slots on both channels are free). On both channels, the sequence of slots repeats endlessly, with slot 0 being marked in a special way so latecomers can detect it. All channels are synchronized by a single global clock.

The protocol supports three classes of traffic: (1) constant data rate connection-oriented traffic, such as uncompressed video, (2) variable data rate connection-oriented traffic, such as file transfer, and (3) datagram traffic, such as UDP packets. For the two connection-oriented protocols, the idea is that for *A* to communicate with *B*, it must first insert a CONNECTION REQUEST frame in a free slot on *B*'s control channel. If *B* accepts, communication can take place on *A*'s data channel.

Each station has two transmitters and two receivers, as follows:

1. A fixed-wavelength receiver for listening to its own control channel.
2. A tunable transmitter for sending on other station's control channel.
3. A fixed-wavelength transmitter for outputting data frames.
4. A tunable receiver for selecting a data transmitter to listen to.

In other words, every station listens to its own control channel for incoming requests but has to tune to the transmitter's wavelength to get the data. Wavelength tuning is done by a Fabry-Perot or Mach-Zehnder interferometer that filters out all wavelengths except the desired wavelength band.

Let us now consider how station *A* sets up a class 2 communication channel with station *B* for, say, file transfer. First, *A* tunes its data receiver to *B*'s data channel and waits for the status slot. This slot tells which control slots are currently assigned and which are free. In Fig. 4-10, for example, we see that of *B*'s eight control slots, 0, 4, and 5 are free. The rest are occupied (indicated by crosses).

A picks one of the free control slots, say, 4, and inserts its CONNECTION REQUEST message there. Since *B* constantly monitors its control channel, it sees the request and grants it by assigning slot 4 to *A*. This assignment is announced in the status slot of the control channel. When *A* sees the announcement, it knows it has a unidirectional connection. If *A* asked for a two-way connection, *B* now repeats the same algorithm with *A*.

It is possible that at the same time *A* tried to grab *B*'s control slot 4, *C* did the same thing. Neither will get it, and both will notice the failure by monitoring the status slot in *B*'s control channel. They now each wait a random amount of time and try again later.

At this point, each party has a conflict-free way to send short control messages to the other one. To perform the file transfer, *A* now sends *B* a control message saying, for example, "Please watch my next data output slot 3. There is a data frame for you in it." When *B* gets the control message, it tunes its receiver to *A*'s output channel to read the data frame. Depending on the higher-layer protocol, *B* can use the same mechanism to send back an acknowledgement if it wishes.

Note that a problem arises if both *A* and *C* have connections to *B* and each of them suddenly tells *B* to look at slot 3. *B* will pick one of these at random, and the other transmission will be lost.

For constant rate traffic, a variation of this protocol is used. When *A* asks for a connection, it simultaneously says something like: Is it all right if I send you a frame in every occurrence of slot 3? If *B* is able to accept (i.e., has no previous commitment for slot 3), a guaranteed bandwidth connection is established. If not, *A* can try again with a different proposal, depending on which output slots it has free.

Class 3 (datagram) traffic uses yet another variation. Instead of writing a CONNECTION REQUEST message into the control slot it just found (4), it writes a DATA FOR YOU IN SLOT 3 message. If *B* is free during the next data slot 3, the transmission will succeed. Otherwise, the data frame is lost. In this manner, no connections are ever needed.

Several variants of the entire protocol are possible. For example, instead of giving each station its own control channel, a single control channel can be shared by all stations. Each station is assigned a block of slots in each group, effectively multiplexing multiple virtual channels onto one physical one.

It is also possible to make do with a single tunable transmitter and a single tunable receiver per station by having each station's channel be divided up into *m* control slots followed by $n + 1$ data slots. The disadvantage here is that senders have to wait longer to capture a control slot and consecutive data frames are further apart because some control information is in the way.

Numerous other WDMA protocols have been proposed, differing in the details. Some have one control channel, some have multiple control channels. Some take propagation delay into account, others do not; some make tuning time an explicit part of the model, others ignore it. The protocols also differ in terms of processing complexity, throughput and scalability. For more information see (Bogineni et al., 1993; Chen, 1994; Chen and Yum, 1991; Jia and Mukherjee, 1993; Levine and Akyildiz, 1995; and Williams et al., 1993).

4.2.6. Wireless LAN Protocols

As the number of portable computing and communication devices grows, so does the demand to connect them to the outside world. Even the very first portable telephones had the ability to connect to other telephones. The first portable computers did not have this capability, but soon afterward, modems became commonplace. To go on-line, these computers had to be plugged into a telephone wall socket. Requiring a wired connection to the fixed network meant that the computers were portable, but not mobile.

To achieve true mobility, portable computers need to use radio (or infrared) signals for communication. In this manner, dedicated users can read and send email while driving or boating. A system of portable computers that communicate by radio can be regarded as a wireless LAN. These LANs have somewhat different properties than conventional LANs and require special MAC sublayer protocols. In this section we will examine some of these protocols. More information about wireless LANs can be found in (Davis and McGuffin, 1995; and Nemzow, 1995).

A common configuration for a wireless LAN is an office building with base stations strategically placed around the building. All the base stations are wired together using copper or fiber. If the transmission power of the base stations and portables is adjusted to have a range of 3 or 4 meters, then each room becomes a single cell, and the entire building becomes a large cellular system, as in the traditional cellular telephony systems we studied in Chap. 2. Unlike cellular telephone systems, each cell has only one channel, covering the entire available bandwidth. Typically its bandwidth is 1 to 2 Mbps.

In our discussions below, we will make the simplifying assumption that all radio transmitters have some fixed range. When a receiver is within range of two active transmitters, the resulting signal will generally be garbled and useless (but with certain exceptions to be discussed later). It is important to realize that in some wireless LANs, not all stations are within range of one another, which leads to a variety of complications. Furthermore, for indoor wireless LANs, the presence of walls between stations can have a major impact on the effective range of each station.

A naive approach to using a wireless LAN might be to try CSMA: just listen for other transmissions and only transmit if no one else is doing so. The trouble is, this protocol is not really appropriate because what matters is interference at the receiver, not at the sender. To see the nature of the problem, consider Fig. 4-11, where four wireless stations are illustrated. For our purposes, it does not matter which are base stations and which are portables. The radio range is such that *A* and *B* are within each other's range and can potentially interfere with one another. *C* can also potentially interfere with both *B* and *D*, but not with *A*.

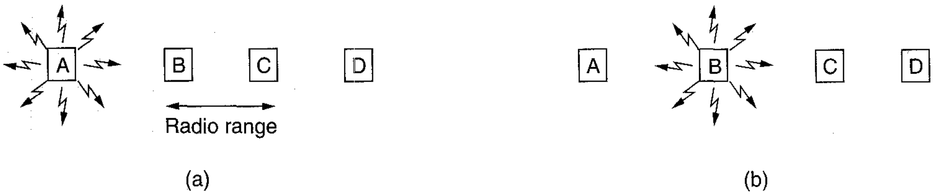


Fig. 4-11. A wireless LAN. (a) *A* transmitting. (b) *B* transmitting.

First consider what happens when *A* is transmitting to *B*, as depicted in Fig. 4-11(a). If *C* senses the medium, it will not hear *A* because *A* is out of range, and thus falsely conclude that it can transmit. If *C* does start transmitting, it will interfere at *B*, wiping out the frame from *A*. The problem of a station not being able to detect a potential competitor for the medium because the competitor is too far away is sometimes called the **hidden station problem**.

Now let us consider the reverse situation: *B* transmitting to *A*, as shown in Fig. 4-11(b). If *C* senses the medium, it will hear an ongoing transmission and falsely conclude that it may not send to *D*, when in fact such a transmission would cause bad reception only in the zone between *B* and *C*, where neither of the intended receivers is located. This situation is sometimes called the **exposed station problem**.

The problem is that before starting a transmission, a station really wants to know whether or not there is activity around the receiver. CSMA merely tells it whether or not there is activity around the station sensing the carrier. With a wire, all signals propagate to all stations so only one transmission can take place at once anywhere in the system. In a system based on short-range radio waves, multiple transmissions can occur simultaneously if they all have different destinations and these destinations are out of range of one another.

Another way to think about this problem is to imagine an office building in which every employee has a wireless portable computer. Suppose that Linda wants to send a message to Milton. Linda's computer senses the local environment and, detecting no activity, starts sending. However, there may still be a collision in Milton's office because a third party may currently be sending to him from a location so far from Linda that her computer could not detect it.

MACA and MACAW

An early protocol designed for wireless LANs is **MACA (Multiple Access with Collision Avoidance)** (Karn, 1990). It was used as the basis for the IEEE 802.11 wireless LAN standard. The basic idea behind it is for the sender to stimulate the receiver into outputting a short frame, so stations nearby can detect this transmission and avoid transmitting themselves for the duration of the upcoming (large) data frame. MACA is illustrated in Fig. 4-12.

Let us consider how *A* sends a frame to *B*. *A* starts by sending an RTS (Request To Send) frame to *B*, as shown in Fig. 4-12(a). This short frame (30 bytes) contains the length of the data frame that will eventually follow. Then *B* replies with a CTS (Clear To Send) frame, as shown in Fig. 4-12(b). The CTS frame contains the data length (copied from the RTS frame). Upon receipt of the CTS frame, *A* begins transmission.

Now let us see how stations overhearing either of these frames react. Any station hearing the RTS is clearly close to *A* and must remain silent long enough for the CTS to be transmitted back to *A* without conflict. Any station hearing the CTS is clearly close to *B* and must remain silent during the upcoming data transmission, whose length it can tell by examining the CTS frame.

In Fig. 4-12, *C* is within range of *A* but not within range of *B*. Therefore it hears the RTS from *A* but not the CTS from *B*. As long as it does not interfere with

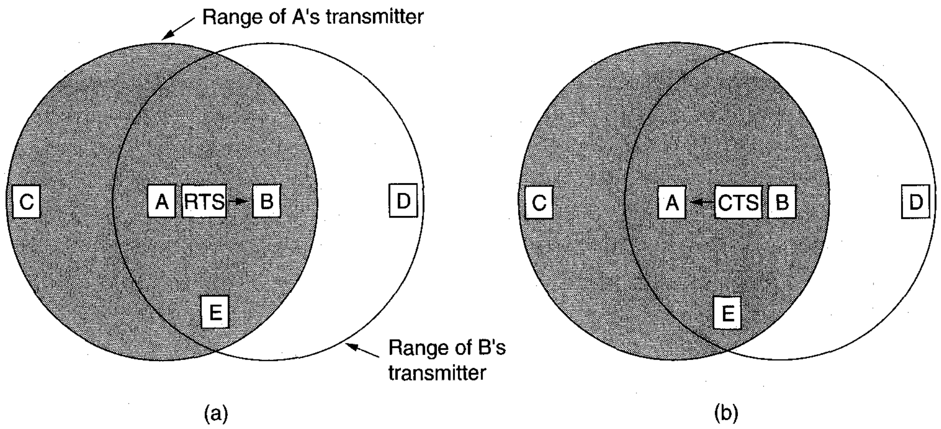


Fig. 4-12. The MACA protocol. (a) *A* sending an RTS to *B*. (b) *B* responding with a CTS to *A*.

the CTS, it is free to transmit while the data frame is being sent. In contrast, *D* is within range of *B* but not *A*. It does not hear the RTS but does hear the CTS. Hearing the CTS tips it off that it is close to a station that is about to receive a frame, so it defers from sending anything until that frame is expected to be finished. Station *E* hears both control messages, and like *D*, must be silent until the data frame is complete.

Despite these precautions, collisions can still occur. For example, *B* and *C* could both send RTS frames to *A* at the same time. These will collide and be lost. In the event of a collision, an unsuccessful transmitter (i.e., one that does not hear a CTS within the expected time interval) waits a random amount of time and tries again later. The algorithm used is binary exponential backoff, which we will study when we come to the IEEE 802.3 LAN.

Based on simulation studies of MACA, Bharghavan et al. (1994) fine tuned MACA to improve its performance and renamed their new protocol **MACAW**. To start with, they noticed that without data link layer acknowledgements, lost frames were not retransmitted until the transport layer noticed their absence, much later. They solved this problem by introducing an ACK frame after each successful data frame. They also observed that CSMA has some utility—namely to keep a station from transmitting an RTS at the same time another nearby station is also doing so to the same destination, so carrier sensing was added. In addition, they decided to run the backoff algorithm separately for each data stream (source-destination pair), rather than for each station. This change improves the fairness of the protocol. Finally, they added a mechanism for stations to exchange information about congestion, and a way to make the backoff algorithm react less violently to temporary problems, to improve system performance.

4.2.7. Digital Cellular Radio

A second form of wireless networking is digital cellular radio, the successor to the AMPS system we studied in Chap. 2. Digital cellular radio presents a somewhat different environment than do wireless LANs and uses different protocols. In particular, it is oriented toward telephony, which requires connections lasting for minutes, rather than milliseconds, so it is more efficient to do channel allocation per call rather than per frame. Nevertheless, the techniques are equally valid for data traffic. In this section we will look at three radically different approaches to channel allocation for wireless digital radio systems, GSM, CDPD, and CDMA.

GSM—Global System for Mobile Communications

The first generation of cellular phones were analog, as described in Chap. 2, but the current generation is digital, using packet radio. Digital transmission has several advantages over analog for mobile communication. First, voice, data, and fax, can be integrated into a single system. Second, as better speech compression algorithms are discovered, less bandwidth will be needed per channel. Third, error-correcting codes can be used to improve transmission quality. Finally, digital signals can be encrypted for security.

Although it might have been nice if the whole world had adopted the same digital standard, such is not the case. The U.S. system, IS-54, and the Japanese system, JDC, have been designed to be compatible with each country's existing analog system, so each AMPS channel could be used either for analog or digital communication.

In contrast, the European digital system, **GSM** (**Global System for Mobile communications**), has been designed from scratch as a fully digital system, without any compromises for the sake of backward compatibility (e.g., having to use the existing frequency slots). Since GSM is also further along than the U.S. system and is currently in use in over 50 countries, inside and outside of Europe, we will use it as an example of digital cellular radio.

GSM was originally designed for use in the 900-MHz band. Later, frequencies were allocated at 1800 MHz, and a second system, closely patterned on GSM, was set up there. The latter is called **DCS 1800**, but it is essentially GSM.

The GSM standard is over 5000 [sic] pages long. A large fraction of this material relates to engineering aspects of the system, especially the design of receivers to handle multipath signal propagation, and synchronizing transmitters and receivers.

A GSM system has up to a maximum of 200 full-duplex channels per cell. Each channel consists of a downlink frequency (from the base station to the mobile stations) and an uplink frequency (from the mobile stations to the base station). Each frequency band is 200 kHz wide as shown in Fig. 4-13.

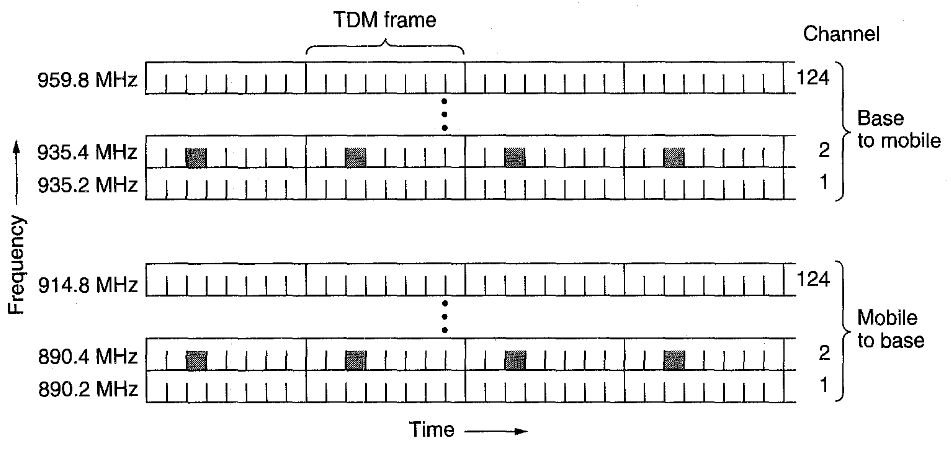


Fig. 4-13. GSM uses 124 frequency channels, each of which use an eight-slot TDM system.

Each of the 124 frequency channels supports eight separate connections using time division multiplexing. Each currently active station is assigned one time slot on one channel. Theoretically, 992 channels can be supported in each cell, but many of them are not available, to avoid frequency conflicts with neighboring cells. In Fig. 4-13, the eight shaded time slots all belong to the same channel, four of them in each direction. If the mobile station assigned to 890.4/935.4 MHz and slot 2 wanted to transmit to the base station, it would use the lower four shaded slots (and the ones following them in time), putting some data in each slot until all the data had been sent.

The TDM slots shown in Fig. 4-13 are part of a complex framing hierarchy. Each TDM slot has a specific structure, and groups of TDM slots form multiframes, also with a specific structure. A simplified version of this hierarchy is shown in Fig. 4-14. Here we can see that each TDM slot consists of a 148-bit data frame. Each data frame starts and ends with three 0 bits, for frame delineation purposes. It also contains two 57-bit *Information* fields, each one having a control bit that indicates whether the following *Information* field is for voice or data. Between the *Information* fields is a 26-bit *Sync* (training) field that is used by the receiver to synchronize to the sender's frame boundaries. A data frame is transmitted in 547 μsec, but a transmitter is only allowed to send one data frame every 4.615 msec, since it is sharing the channel with seven other stations. The gross rate of each channel is 270,833 bps, divided among eight users. Discounting all the overhead, each connection can send one compressed voice signal or 9600 bps of data.

As can be seen from Fig. 4-14, eight data frames make up a TDM frame, and 26 TDM frames make up a 120-msec multiframe. Of the 26 TDM frames in a

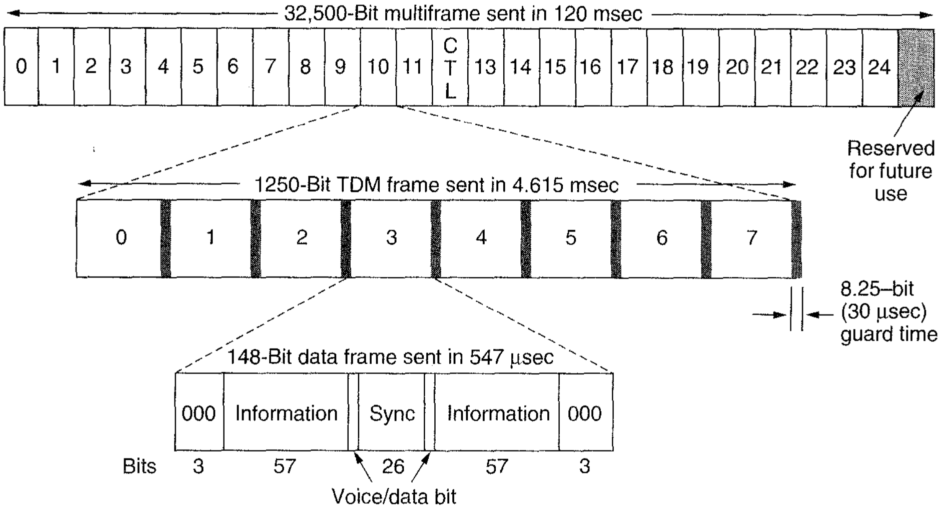

32,500-Bit multiframe sent in 120 msec
0
1
2
3
4
5
6
7
8
9
10
11
C T L
13
14
15
16
17
18
19
20
21
22
23
24
Reserved for future use
1250-Bit TDM frame sent in 4.615 msec
0
1
2
3
4
5
6
7
8.25–bit (30 μsec) guard time
148-Bit data frame sent in 547 μsec
000
Information
Sync
Information
000
Bits
3
57
26
57
3
Voice/data bit

Fig. 4-14. A portion of the GSM framing structure.

multiframe, slot 12 is used for control and slot 25 is reserved for future use, so only 24 are available for user traffic.

However, in addition to the 26-slot multiframe shown in Fig. 4-14, a 51-slot multiframe (not shown) is also used. Some of these slots are used to hold several control channels used to manage the system. The **broadcast control channel** is a continuous stream of output from the base station containing its identity and the channel status. All mobile stations monitor its signal strength to see when they have moved into a new cell.

The **dedicated control channel** is used for location updating, registration, and call setup. In particular, each base station maintains a database of mobile stations currently under its jurisdiction. Information needed to maintain this database is sent on the dedicated control channel.

Finally, there is the **common control channel**, which is split up into three logical subchannels. The first of these subchannels is the **paging channel**, which the base station uses to announce incoming calls. Each mobile station monitors it continuously to watch for calls it should answer. The second is the **random access channel**, which runs a slotted ALOHA system to allow a mobile station to request a slot on the dedicated control channel. Using this slot, the station can set up a call. The assigned slot is announced on the third subchannel, the **access grant channel**.

All in all, GSM is a fairly complex system. It handles channel access using a combination of slotted ALOHA, FDM and TDM. For more information about GSM, including aspects of the system that we have not discussed, for example, the protocol layering architecture, see (Rahnema, 1993).

CDPD—Cellular Digital Packet Data

GSM is basically circuit switched. A mobile computer with a special modem can place a call using a GSM telephone the same way it would place one on a hardwired telephone. However, using this strategy is not without problems. For one, handoffs between base stations are frequent, sometimes even with stationary users (base stations can shuffle users around for load balancing), and each handoff results in losing ca. 300 msec of data. For another, GSM can suffer from a high error rate. Typing an "a" and having it echoed as an "m" gets tiresome quickly. Finally, wireless calls are expensive, and costs mount quickly because the charge is per minute of connect time, not per byte sent.

One approach to solving these problems is a packet-switched digital datagram service called **CDPD** (**Cellular Digital Packet Data**). It is built on top of AMPS (see Chap. 2) and entirely compatible with AMPS. Basically, any idle 30-kHz channel can be temporarily grabbed for sending data frames at a gross rate of 19.2 kbps. Because CDPD involves quite a bit of overhead, the net data rate is closer to 9600 bps. Still, a connectionless, wireless datagram system for sending, for example, IP packets, using the existing cellular phone system is an interesting proposition for many users, so its use is growing rapidly.

CDPD follows the OSI model closely. The physical layer deals with the details of modulation and radio transmission, which do not concern us here. Data link, network, and transport protocols also exist but are not of special interest to us either. Instead, we will give a general description of the system and then describe the medium access protocol. For more information about the full CDPD system, see (Quick and Balachandran, 1993).

A CDPD system consists of three kinds of stations: mobile hosts, base stations, and base interface stations (in CDPD jargon: mobile end systems, mobile data base systems, and mobile data intermediate systems, respectively). These stations interact with stationary hosts and standard routers, of the kind found in any WAN. The mobile hosts are the users' portable computers. The base stations are the transmitters that talk to the mobile hosts. The base interface stations are special nodes that interface all the base stations in a CDPD provider's area to a standard (fixed) router for further transmission through the Internet or other WAN. This arrangement is shown in Fig. 4-15.

Three kinds of interfaces are defined in CDPD. The **E-interface** (external to the CDPD provider) connects a CDPD area to a fixed network. This interface must be well defined to allow CDPD to connect to a variety of networks. The **I-interface** (internal to the CDPD provider) connects two CDPD areas together. It must be standardized to allow users to roam between areas. The third one is the **A-interface**, (air interface) between the base station and mobile hosts. This is the most interesting one, so we will now examine it more closely.

Data over the air interface are sent using compression, encryption, and error correction. Units of 274 compressed, encrypted data bits are wrapped in 378-bit

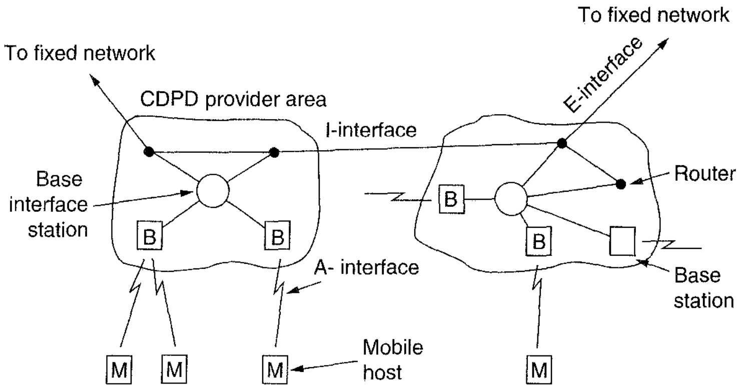


Fig. 4-15. An example CDPD system.

blocks using a Reed-Solomon error correcting code. To each RS block is added seven 6-bit flag words, to form a total of 420-bit blocks. Each 420-bit block is divided up into seven 60-bit microblocks, which are sent consecutively. Each microblock has its own 6-bit flag word, used for indicating channel status. These microblocks go over a 19.2-kbps downlink channel (from the base) or over a second 19.2-kbps uplink channel (to the base), in full-duplex mode. In effect, both the downlink and uplink channel are slotted in time, as a sequence of 60-bit microblocks. Each microblock lasts for 3.125 msec.

Each CDPD cell has only one downlink/uplink pair available for data. The downlink channel is easy to manage since there is only one sender per cell: the base station. All frames sent on it are broadcast, with each mobile host selecting out those destined for it or for everyone.

The tricky part is the uplink channel, for which all mobile hosts wishing to send must contend. When a mobile host has a frame to send, it watches the downlink channel for a flag bit telling whether the current uplink slot is busy or idle. If it is busy, instead of just waiting for the next time slot, it skips a random number of slots and tries again. If it again sees that the uplink channel is busy, it waits a longer random time, and repeats the procedure. The statistically average waiting time doubles with each unsuccessful attempt. When it finally finds the channel supposedly idle, it begins transmitting its microblock.

The point of this algorithm, called **DSMA** (**Digital Sense Multiple Access**), is to prevent all the mobile hosts from jumping on the uplink channel as soon it goes idle. It somewhat resembles the slotted p-persistent CSMA protocol we mentioned earlier, since it, too, uses discrete time slots on both channels.

The trouble is, despite DSMA, a collision with another mobile host is still possible, since two or more of them may pick the same time slot to starting sending. To allow mobile hosts to discover whether or not they have suffered a collision, a flag bit in each microblock tells whether a previous microblock on the uplink channel was received correctly. Unfortunately, the base station cannot make the determination instantly after a microblock terminates, so the correct/incorrect reception of microblock *n* is delayed until microblock $n + 2$.

Since it cannot tell if its transmission was successful, if a sender has more microblocks to send, it just goes ahead, without having to reacquire the channel. If in the *following* time slot it sees that its *previous* transmission failed, it stops. Otherwise it continues transmitting, up to a certain maximum number of Reed-Solomon blocks, or until the base station sets a flag bit on the downlink channel to indicate that it has heard enough from this particular sender for the moment.

An additional property of CDPD is that data users are second-class citizens. When a new voice call is about to be assigned to a channel currently in use for CDPD, the base station sends a special signal on the downlink, closing down the channel. If the base station already knows the number of the new CDPD channel, it announces it. Otherwise, mobile hosts have to hunt around among a designated set of potential CDPD channels to find it. In this way, CDPD can suck up any idle capacity in a cell, without interfering with the big cash cow, voice.

It should be clear from this description that CDPD was added to the voice system after the latter was already operational, and that its design was subject to the constraint that no changes could be made to the existing voice system. Consequently, when channel selection for voice calls occurs, the algorithm is not aware of the existence of CDPD. This is the reason that the CDPD channel is sometimes suddenly preempted. However, nothing in the design prevents having dedicated CDPD channels. As CDPD grows in popularity, providers are likely to reserve channels exclusively for it.

CDMA—Code Division Multiple Access

GSM might be described as a brute force solution to channel allocation. It uses a combination of practically every known technique (ALOHA, TDM, FDM) intertwined in complex ways. CDPD for single-frame transmissions is fundamentally nonpersistent CSMA. Now we will examine yet another method for allocating a wireless channel, **CDMA** (**Code Division Multiple Access**).

CDMA is completely different from all the other allocation techniques we have studied so far. Some of these have been based on dividing the channel into frequency bands and assigning those statically (FDM) or on demand (wavelength division multiplexing), with the owner using the band indefinitely. Others allocate the channel in bursts, giving stations the entire channel statically (TDM with fixed time slots) or dynamically (ALOHA). CDMA allows each station to transmit over the entire frequency spectrum all the time. Multiple simultaneous transmissions are separated using coding theory. CDMA also relaxes the assumption that colliding frames are totally garbled. Instead, it assumes that multiple signals add linearly.

Before getting into the algorithm, let us consider the cocktail party theory of channel access. In a large room, many pairs of people are conversing. TDM is when all the people are in the middle of the room, but they take turns speaking, first one then another. FDM is when the people group into widely separated clumps, each clump holding its own conversation at the same time as, but still independent of, the others. CDMA is when they are all in the middle of the room talking at once, but with each pair in a different language. The French-speaking couple just hones in on the French, rejecting everything else as noise. Thus the key to CDMA is to be able to extract the desired signal while rejecting everything else as random noise.

In CDMA each bit time is subdivided into *m* short intervals called **chips**. Typically there are 64 or 128 chips per bit, but in the example given below we will use 8 chips/bit for simplicity.

Each station is assigned a unique *m*-bit code or **chip sequence**. To transmit a 1 bit, a station sends its chip sequence. To transmit a 0 bit, it sends the one's complement of its chip sequence. No other patterns are permitted. Thus for $m = 8$, if station *A* is assigned the chip sequence 00011011, it sends a 1 bit by sending 00011011 and a 0 bit by sending 11100100.

Increasing the amount of information to be sent from *b* bits/sec to *mb* chips/sec can only be done if the bandwidth available is increased by a factor of *m*, making CDMA a form of spread spectrum communication (assuming no changes in the modulation or encoding techniques). If we have a 1-MHz band available for 100 stations, with FDM each one would have 10 kHz and could send at 10 kbps (assuming 1 bit per Hz). With CDMA, each station uses the full 1 MHz, so the chip rate is 1 megachip per second. With fewer than 100 chips per bit, the effective bandwidth per station is higher for CDMA than FDM, and the channel allocation problem is also solved, as we will see shortly.

For pedagogical purposes, it is more convenient to use a bipolar notation, with binary 0 being −1 and binary 1 being +1. We will show chip sequences in parentheses, so a 1 bit for station *A* now becomes (−1 −1 −1 +1 +1 −1 +1 +1). In Fig. 4-16(a) we show the binary chip sequences assigned to four example stations. In Fig. 4-16(b) we show them in our bipolar notation.

Each station has its own unique chip sequence. Let us use the symbol **S** to indicate the *m*-chip vector for station *S*, and $\overline{\mathbf{S}}$ for its negation. All chip sequences are pairwise **orthogonal**, by which we mean that the normalized inner product of any two distinct chip sequences, **S** and **T** (written as **S•T**) is 0. In mathematical terms,

$$\mathbf{S}\bullet\mathbf{T} \equiv \frac{1}{m}\sum_{i=1}^{m} S_i T_i = 0 \tag{4-5}$$

In plain English, as many pairs are the same as are different. This orthogonality property will prove crucial later on. Note that if $\mathbf{S}\bullet\mathbf{T} = 0$ then $\mathbf{S}\bullet\overline{\mathbf{T}}$ is also 0. The normalized inner product of any chip sequence with itself is 1:

$$\mathbf{S}\bullet\mathbf{S} = \frac{1}{m}\sum_{i=1}^{m} S_i S_i = \frac{1}{m}\sum_{i=1}^{m} S_i^2 = \frac{1}{m}\sum_{i=1}^{m} (\pm 1)^2 = 1$$

(a)	(b)
A: 0 0 0 1 1 0 1 1	A: (−1 −1 −1 +1 +1 −1 +1 +1)
B: 0 0 1 0 1 1 1 0	B: (−1 −1 +1 −1 +1 +1 +1 −1)
C: 0 1 0 1 1 1 0 0	C: (−1 +1 −1 +1 +1 +1 −1 −1)
D: 0 1 0 0 0 0 1 0	D: (−1 +1 −1 −1 −1 −1 +1 −1)

Six examples:

– – 1 –	**C**	S_1 = (−1 +1 −1 +1 +1 +1 −1 −1)
– 1 1 –	**B + C**	S_2 = (−2 0 0 0 +2 +2 0 −2)
1 0 – –	**A + B**	S_3 = (0 0 −2 +2 0 −2 0 +2)
1 0 1 –	**A + B + C**	S_4 = (−1 +1 −3 +3 −1 −1 −1 +1)
1 1 1 1	**A + B + C + D**	S_5 = (−4 0 −2 0 +2 0 +2 −2)
1 1 0 1	**A + B + $\bar{\text{C}}$ + D**	S_6 = (−2 −2 0 −2 0 −2 +4 0)

(c)

$S_1 \bullet C = (1 +1 +1 +1 +1 +1 +1 +1)/8 = 1$

$S_2 \bullet C = (2 +0 +0 +0 +2 +2 +0 +2)/8 = 1$

$S_3 \bullet C = (0 +0 +2 +2 +0 -2 +0 -2)/8 = 0$

$S_4 \bullet C = (1 +1 +3 +3 +1 -1 +1 -1)/8 = 1$

$S_5 \bullet C = (4 +0 +2 +0 +2 +0 -2 +2)/8 = 1$

$S_6 \bullet C = (2 -2 +0 -2 +0 -2 -4 +0)/8 = -1$

(d)

Fig. 4-16. (a) Binary chip sequences for four stations. (b) Bipolar chip sequences. (c) Six examples of transmissions. (d) Recovery of station C's signal.

This follows because each of the m terms in the inner product is 1, so the sum is m. Also note that $\mathbf{S} \bullet \overline{\mathbf{S}} = -1$.

During each bit time, a station can transmit a 1 by sending its chip sequence, it can transmit a 0 by sending the negative of its chip sequence, or it can be silent and transmit nothing. For the moment, we assume that all stations are synchronized in time, so all chip sequences begin at the same instant.

When two or more stations transmit simultaneously, their bipolar signals add linearly. For example, if in one chip period three stations output +1 and one station outputs −1, the result is +2. One can think of this as adding voltages: three stations outputting +1 volts and 1 station outputting −1 volts gives 2 volts.

In Fig. 4-16(c) we see six examples of one or more stations transmitting at the same time. In the first example, C transmits a 1 bit, so we just get C's chip sequence. In the second example, both B and C transmit 1 bits, so we get the sum of their bipolar chip sequences, namely:

$$(-1\ -1\ +1\ -1\ +1\ +1\ +1\ -1) + (-1\ +1\ -1\ +1\ +1\ +1\ -1\ -1) = (-2\ \ 0\ \ 0\ \ 0\ +2\ +2\ \ 0\ -2)$$

In the third example, station A sends a 1 and station B sends a 0. The others are silent. In the fourth example, A and C send a 1 bit while B sends a 0 bit. In the fifth example, all four stations send a 1 bit. Finally, in the last example, A, B, and

D send a 1 bit, while *C* sends a 0 bit. Note that each of the six sequences S_1 through S_6 given in Fig. 4-16(c) represents only one bit time.

To recover the bit stream of an individual station, the receiver must know that station's chip sequences in advance. It does the recovery by computing the normalized inner product of the received chip sequence (the linear sum of all the stations that transmitted) and the chip sequence of the station whose bit stream it is trying to recover. If the received chip sequence is **S** and the receiver is trying to listen to a station whose chip sequence is **C**, it just computes the normalized inner product, $\mathbf{S} \bullet \mathbf{C}$.

To see why this works, imagine that two stations, *A* and *C*, both transmit a 1 bit at the same time that *B* transmits a 0 bit. The receiver sees the sum: $\mathbf{S} = \mathbf{A} + \overline{\mathbf{B}} + \mathbf{C}$ and computes

$$\mathbf{S} \bullet \mathbf{C} = (\mathbf{A} + \overline{\mathbf{B}} + \mathbf{C}) \bullet \mathbf{C} = \mathbf{A} \bullet \mathbf{C} + \overline{\mathbf{B}} \bullet \mathbf{C} + \mathbf{C} \bullet \mathbf{C} = 0 + 0 + 1 = 1$$

The first two terms vanish because all pairs of chip sequences have been carefully chosen to be orthogonal, as shown in Eq. (4-5). Now it should be clear why this property must be imposed on the chip sequences.

An alternative way of thinking about this situation is to imagine that the three chip sequences all came in separately, rather than summed. Then the receiver would compute the inner product with each one separately and add the results. Due to the orthogonality property, all the inner products except $\mathbf{C} \bullet \mathbf{C}$ would be 0. Adding them and then doing the inner product is in fact the same as doing the inner products and then adding those.

To make the decoding process more concrete, let us consider the six examples of Fig. 4-16(d) again. Suppose that the receiver is interested in extracting the bit sent by station *C* from each of the six sums S_1 through S_6. It calculates the bit by summing the pairwise products of the received **S** and the **C** vector of Fig. 4-16(b), and then taking 1/8 of the result (since $m = 8$ here). As shown, each time the correct bit is decoded. It is just like speaking French.

In an ideal, noiseless CDMA system, the capacity (i.e., number of stations) can be made arbitrarily large, just as the capacity of a noiseless Nyquist channel can be made arbitrarily large by using more and more bits per sample. In practice, physical limitations reduce the capacity considerably. First, we have assumed that all the chips are synchronized in time. In reality, doing so is impossible. What can be done is that the sender and receiver synchronize by having the sender transmit a long enough known chip sequence that the receiver can lock onto. All the other (unsynchronized) transmissions are then seen as random noise. If there are not too many of them, however, the basic decoding algorithm still works fairly well. A large body of theory exists relating the superposition of chip sequences to noise level (Pickholtz et al., 1982). As one might expect, the longer the chip sequence, the higher the probability of detecting it correctly in the presence of noise. For extra security, the bit sequence can use an error correcting code. Chip sequences never use error correcting codes.

An implicit assumption in the above discussion is that the power levels of all stations are the same as perceived by the receiver. CDMA is typically used for wireless systems with a fixed base station and many mobile stations at varying distances from it. The power levels received at the base station depend on how far away the transmitters are. A good heuristic here is for each mobile station to transmit to the base station at the inverse of the power level it receives from the base station, so a mobile station receiving a weak signal from the base will use more power than one getting a strong signal. The base station can also give explicit commands to the mobile stations to increase or decrease their transmission power.

We have also assumed that the receiver knows who the sender is. In principle, given enough computing capacity, the receiver can listen to all the senders at once by running the decoding algorithm for each of them in parallel. In real life, suffice it to say that this is easier said than done. CDMA also has many other complicating factors that have been glossed over in this brief introduction. Nevertheless, CDMA is a clever scheme that is being rapidly introduced for wireless mobile communication.

Readers with a solid electrical engineering background who want to gain a deeper understanding of CDMA should read (Viterbi, 1995). An alternative spreading scheme, in which the spreading is over time rather than frequency, is described in (Crespo et al., 1995).

4.3. IEEE STANDARD 802 FOR LANS AND MANS

We have now finished our general discussion of abstract channel allocation protocols, so it is time to see how these principles apply to real systems, in particular, LANs. As discussed in Sec. 1.7.2, IEEE has produced several standards for LANs. These standards, collectively known as **IEEE 802**, include CSMA/CD, token bus, and token ring. The various standards differ at the physical layer and MAC sublayer but are compatible at the data link layer. The IEEE 802 standards have been adopted by ANSI as American National Standards, by NIST as government standards, and by ISO as international standards (known as ISO 8802). They are surprisingly readable (as standards go).

The standards are divided into parts, each published as a separate book. The 802.1 standard gives an introduction to the set of standards and defines the interface primitives. The 802.2 standard describes the upper part of the data link layer, which uses the **LLC** (**Logical Link Control**) protocol. Parts 802.3 through 802.5 describe the three LAN standards, the CSMA/CD, token bus, and token ring standards, respectively. Each standard covers the physical layer and MAC sublayer protocol. The next three sections cover these three systems. Additional information can be found in (Stallings, 1993b).

4.3.1. IEEE Standard 802.3 and Ethernet

The IEEE 802.3 standard is for a 1-persistent CSMA/CD LAN. To review the idea, when a station wants to transmit, it listens to the cable. If the cable is busy, the station waits until it goes idle; otherwise it transmits immediately. If two or more stations simultaneously begin transmitting on an idle cable, they will collide. All colliding stations then terminate their transmission, wait a random time, and repeat the whole process all over again.

The 802.3 standard has an interesting history. The real beginning was the ALOHA system constructed to allow radio communication between machines scattered over the Hawaiian Islands. Later, carrier sensing was added, and Xerox PARC built a 2.94-Mbps CSMA/CD system to connect over 100 personal workstations on a 1-km cable (Metcalfe and Boggs, 1976). This system was called **Ethernet** after the *luminiferous ether*, through which electromagnetic radiation was once thought to propagate. (When the Nineteenth Century British physicist James Clerk Maxwell discovered that electromagnetic radiation could be described by a wave equation, scientists assumed that space must be filled with some ethereal medium in which the radiation was propagating. Only after the famous Michelson-Morley experiment in 1887, did physicists discover that electromagnetic radiation could propagate in a vacuum.)

The Xerox Ethernet was so successful that Xerox, DEC, and Intel drew up a standard for a 10-Mbps Ethernet. This standard formed the basis for 802.3. The published 802.3 standard differs from the Ethernet specification in that it describes a whole family of 1-persistent CSMA/CD systems, running at speeds from 1 to 10-Mbps on various media. Also, the one header field differs between the two (the 802.3 length field is used for packet type in Ethernet). The initial standard also gives the parameters for a 10 Mbps baseband system using 50-ohm coaxial cable. Parameter sets for other media and speeds came later.

Many people (incorrectly) use the name "Ethernet" in a generic sense to refer to all CSMA/CD protocols, even though it really refers to a specific product that almost implements 802.3. We will use the terms "802.3" and "CSMA/CD" except when specifically referring to the Ethernet product in the next few paragraphs.

802.3 Cabling

Since the name "Ethernet" refers to the cable (the ether), let us start our discussion there. Four types of cabling are commonly used, as shown in Fig. 4-17. Historically, **10Base5** cabling, popularly called **thick Ethernet,** came first. It resembles a yellow garden hose, with markings every 2.5 meters to show where the taps go. (The 802.3 standard does not actually *require* the cable to be yellow, but it does *suggest* it.) Connections to it are generally made using **vampire taps**, in which a pin is carefully forced halfway into the coaxial cable's core. The notation 10Base5 means that it operates at 10 Mbps, uses baseband signaling, and can support segments of up to 500 meters.

Name	Cable	Max. segment	Nodes/seg.	Advantages
10Base5	Thick coax	500 m	100	Good for backbones
10Base2	Thin coax	200 m	30	Cheapest system
10Base-T	Twisted pair	100 m	1024	Easy maintenance
10Base-F	Fiber optics	2000 m	1024	Best between buildings

Fig. 4-17. The most common kinds of baseband 802.3 LANs.

Historically, the second cable type was **10Base2** or **thin Ethernet,** which, in contrast to the garden-hose-like thick Ethernet, bends easily. Connections to it are made using industry standard BNC connectors to form T junctions, rather than using vampire taps. These are easier to use and more reliable. Thin Ethernet is much cheaper and easier to install, but it can run for only 200 meters and can handle only 30 machines per cable segment.

Detecting cable breaks, bad taps, or loose connectors can be a major problem with both media. For this reason, techniques have been developed to track them down. Basically, a pulse of known shape is injected into the cable. If the pulse hits an obstacle or the end of the cable, an echo will be generated and sent back. By carefully timing the interval between sending the pulse and receiving the echo, it is possible to localize the origin of the echo. This technique is called **time domain reflectometry**.

The problems associated with finding cable breaks have driven systems toward a different kind of wiring pattern, in which all stations have a cable running to a central **hub**. Usually, these wires are telephone company twisted pairs, since most office buildings are already wired this way, and there are normally plenty of spare pairs available. This scheme is called **10Base-T**.

These three wiring schemes are illustrated in Fig. 4-18. For 10Base5, a **transceiver** is clamped securely around the cable so that its tap makes contact with the inner core. The transceiver contains the electronics that handle carrier detection and collision detection. When a collision is detected, the transceiver also puts a special invalid signal on the cable to ensure that all other transceivers also realize that a collision has occurred.

With 10Base5, a **transceiver cable** connects the transceiver to an interface board in the computer. The transceiver cable may be up to 50 meters long and contains five individually shielded twisted pairs. Two of the pairs are for data in and data out, respectively. Two more are for control signals in and out. The fifth pair, which is not always used, allows the computer to power the transceiver electronics. Some transceivers allow up to eight nearby computers to be attached to them, to reduce the number of transceivers needed.

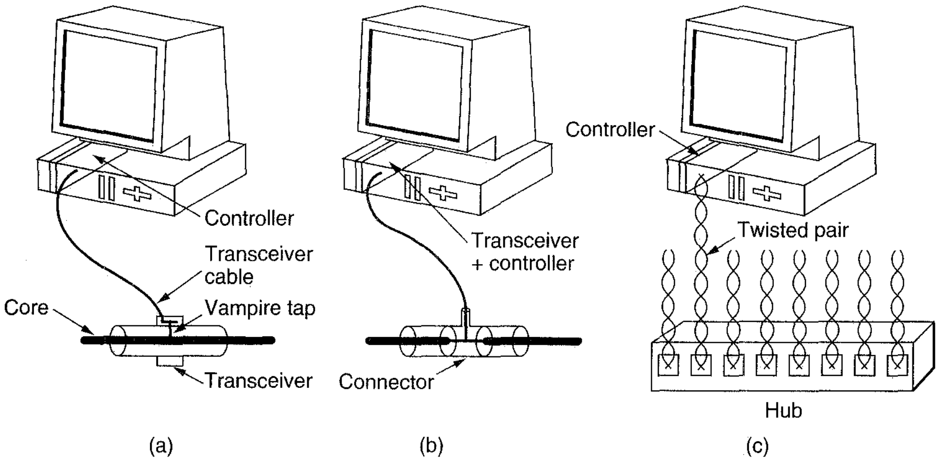


Fig. 4-18. Three kinds of 802.3 cabling. (a) 10Base5. (b) 10Base2. (c) 10Base-T.

The transceiver cable terminates on an interface board inside the computer. The interface board contains a controller chip that transmits frames to, and receives frames from, the transceiver. The controller is responsible for assembling the data into the proper frame format, as well as computing checksums on outgoing frames and verifying them on incoming frames. Some controller chips also manage a pool of buffers for incoming frames, a queue of buffers to be transmitted, DMA transfers with the host computers, and other aspects of network management.

With 10Base2, the connection to the cable is just a passive BNC T-junction connector. The transceiver electronics are on the controller board, and each station always has its own transceiver.

With 10Base-T, there is no cable at all, just the hub (a box full of electronics). Adding or removing a station is simpler in this configuration, and cable breaks can be detected easily. The disadvantage of 10Base-T is that the maximum cable run from the hub is only 100 meters, maybe 150 meters if high-quality (category 5) twisted pairs are used. Also, a large hub costs thousands of dollars. Still, 10Base-T is becoming steadily more popular due to the ease of maintenance that it offers. A faster version of 10Base-T (100Base-T) will be discussed later in this chapter.

A fourth cabling option for 802.3 is **10Base-F**, which uses fiber optics. This alternative is expensive due to the cost of the connectors and terminators, but it has excellent noise immunity and is the method of choice when running between buildings or widely separated hubs.

Figure 4-19 shows different ways of wiring up a building. In Fig. 4-19(a), a single cable is snaked from room to room, with each station tapping onto it at the nearest point. In Fig. 4-19(b), a vertical spine runs from the basement to the roof, with horizontal cables on each floor connected to it by special amplifiers (repeaters). In some buildings the horizontal cables are thin, and the backbone is thick. The most general topology is the tree, as in Fig. 4-19(c), because a network with two paths between some pairs of stations would suffer from interference between the two signals.

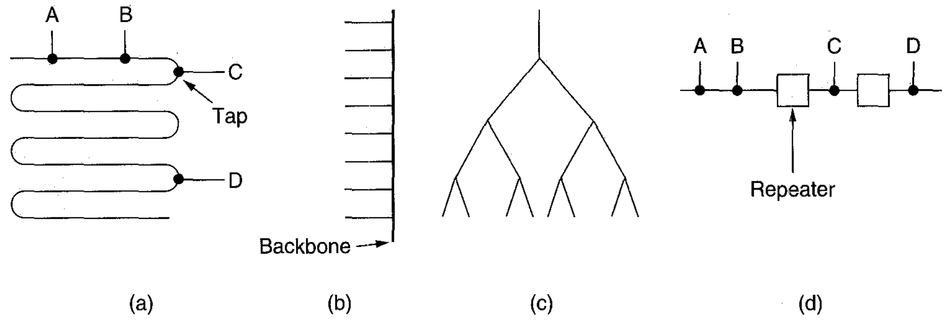


Fig. 4-19. Cable topologies. (a) Linear. (b) Spine. (c) Tree. (d) Segmented.

Each version of 802.3 has a maximum cable length per segment. To allow larger networks, multiple cables can be connected by **repeaters**, as shown in Fig. 4-19(d). A repeater is a physical layer device. It receives, amplifies, and retransmits signals in both directions. As far as the software is concerned, a series of cable segments connected by repeaters is no different than a single cable (except for some delay introduced by the repeaters). A system may contain multiple cable segments and multiple repeaters, but no two transceivers may be more than 2.5 km apart and no path between any two transceivers may traverse more than four repeaters.

Manchester Encoding

None of the versions of 802.3 use straight binary encoding with 0 volts for a 0 bit and 5 volts for a 1 bit because it leads to ambiguities. If one station sends the bit string 0001000, others might falsely interpret it as 10000000 or 01000000 because they cannot tell the difference between an idle sender (0 volts) and a 0 bit (0 volts).

What is needed is a way for receivers to unambiguously determine the start, end, or middle of each bit without reference to an external clock. Two such approaches are called **Manchester encoding** and **differential Manchester encoding**. With Manchester encoding, each bit period is divided into two equal intervals. A binary 1 bit is sent by having the voltage set high during the first interval and low in the second one. A binary 0 is just the reverse: first low and then high. This scheme ensures that every bit period has a transition in the middle, making it easy for the receiver to synchronize with the sender. A disadvantage of Manchester encoding is that it requires twice as much bandwidth as straight binary encoding, because the pulses are half the width. Manchester encoding is shown in Fig. 4-20(b).

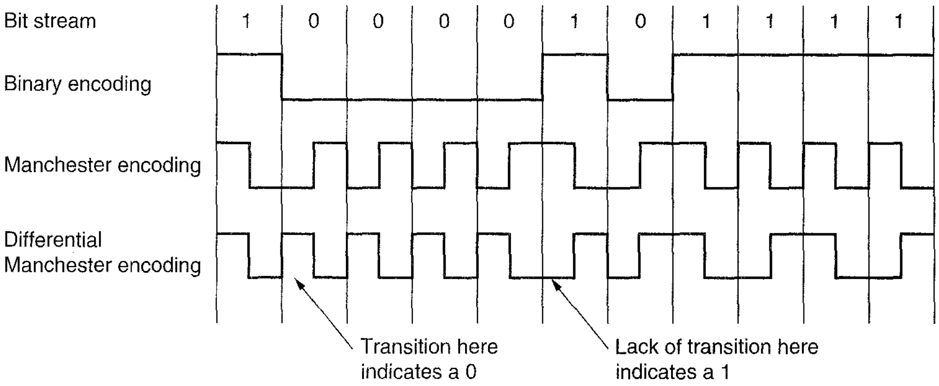

Bit stream
1
0
0
0
0
1
0
1
1
1
1
Binary encoding
Manchester encoding
Differential Manchester encoding
Transition here indicates a 0
Lack of transition here indicates a 1

Fig. 4-20. (a) Binary encoding. (b) Manchester encoding. (c) Differential Manchester encoding.

Differential Manchester encoding, shown in Fig. 4-20(c), is a variation of basic Manchester encoding. In it, a 1 bit is indicated by the absence of a transition at the start of the interval. A 0 bit is indicated by the presence of a transition at the start of the interval. In both cases, there is a transition in the middle as well. The differential scheme requires more complex equipment but offers better noise immunity. All 802.3 baseband systems use Manchester encoding due to its simplicity. The high signal is +0.85 volts and the low signal is −0.85 volts, giving a DC value of 0 volts.

The 802.3 MAC Sublayer Protocol

The 802.3 (IEEE, 1985a) frame structure is shown in Fig. 4-21. Each frame starts with a *Preamble* of 7 bytes, each containing the bit pattern 10101010. The Manchester encoding of this pattern produces a 10-MHz square wave for 5.6 μsec to allow the receiver's clock to synchronize with the sender's. Next comes a *Start of frame* byte containing 10101011 to denote the start of the frame itself.

The frame contains two addresses, one for the destination and one for the source. The standard allows 2-byte and 6-byte addresses, but the parameters defined for the 10-Mbps baseband standard use only the 6-byte addresses. The high-order bit of the destination address is a 0 for ordinary addresses and 1 for group addresses. Group addresses allow multiple stations to listen to a single address. When a frame is sent to a group address, all the stations in the group receive it. Sending to a group of stations is called **multicast**. The address consisting of all 1 bits is reserved for **broadcast**. A frame containing all 1s in the destination field is delivered to all stations on the network.

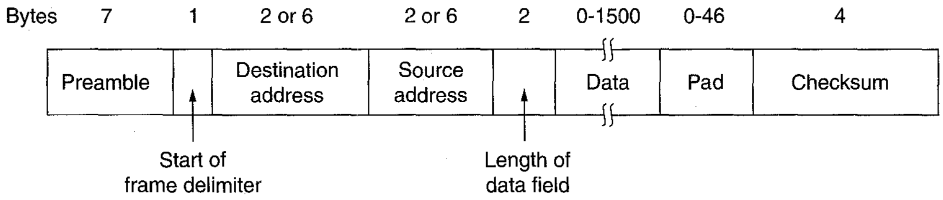


Fig. 4-21. The 802.3 frame format.

Another interesting feature of the addressing is the use of bit 46 (adjacent to the high-order bit) to distinguish local from global addresses. Local addresses are assigned by each network administrator and have no significance outside the local network. Global addresses, in contrast, are assigned by IEEE to ensure that no two stations anywhere in the world have the same global address. With $48 - 2 = 46$ bits available, there are about 7×10^{13} global addresses. The idea is that any station can uniquely address any other station by just giving the right 48-bit number. It is up to the network layer to figure out how to locate the destination.

The *Length* field tells how many bytes are present in the data field, from a minimum of 0 to a maximum of 1500. While a data field of 0 bytes is legal, it causes a problem. When a transceiver detects a collision, it truncates the current frame, which means that stray bits and pieces of frames appear on the cable all the time. To make it easier to distinguish valid frames from garbage, 802.3 states that valid frames must be at least 64 bytes long, from destination address to checksum. If the data portion of a frame is less than 46 bytes, the pad field is used to fill out the frame to the minimum size.

Another (and more important) reason for having a minimum length frame is to prevent a station from completing the transmission of a short frame before the first bit has even reached the far end of the cable, where it may collide with another frame. This problem is illustrated in Fig. 4-22. At time 0, station *A*, at one end of the network, sends off a frame. Let us call the propagation time for this frame to reach the other end τ. Just before the frame gets to the other end (i.e., at time $\tau - \varepsilon$) the most distant station, *B*, starts transmitting. When *B* detects that it is receiving more power than it is putting out, it knows that a collision has occurred, so it aborts its transmission and generates a 48-bit noise burst to warn all other stations. At about time 2τ, the sender sees the noise burst and aborts its transmission, too. It then waits a random time before trying again.

If a station tries to transmit a very short frame, it is conceivable that a collision occurs, but the transmission completes before the noise burst gets back at 2τ. The sender will then incorrectly conclude that the frame was successfully sent. To prevent this situation from occurring, all frames must take more than 2τ to send. For a 10-Mbps LAN with a maximum length of 2500 meters and four

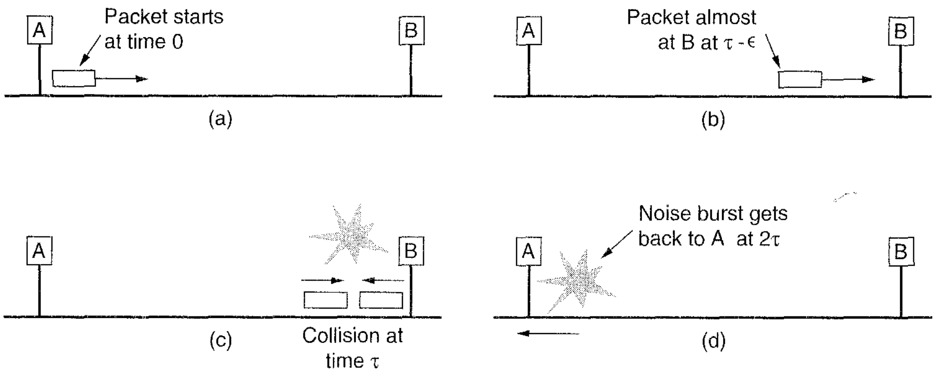


Fig. 4-22. Collision detection can take as long as 2τ.

repeaters (from the 802.3 specification), the minimum allowed frame must take 51.2 μsec. This time corresponds to 64 bytes. Frames with fewer bytes are padded out to 64 bytes.

As the network speed goes up, the minimum frame length must go up or the maximum cable length must come down, proportionally. For a 2500-meter LAN operating at 1 Gbps, the minimum frame size would have to be 6400 bytes. Alternatively, the minimum frame size could be 640 bytes and the maximum distance between any two stations 250 meters. These restrictions are becoming increasingly painful as we move toward gigabit networks.

The final 802.3 field is the *Checksum*. It is effectively a 32-bit hash code of the data. If some data bits are erroneously received (due to noise on the cable), the checksum will almost certainly be wrong, and the error will be detected. The checksum algorithm is a cyclic redundancy check of the kind discussed in Chap. 3.

The Binary Exponential Backoff Algorithm

Let us now see how randomization is done when a collision occurs. The model is that of Fig. 4-5. After a collision, time is divided up into discrete slots whose length is equal to the worst case round-trip propagation time on the ether (2τ). To accommodate the longest path allowed by 802.3 (2.5 km and four repeaters), the slot time has been set to 512 bit times, or 51.2 μsec.

After the first collision, each station waits either 0 or 1 slot times before trying again. If two stations collide and each one picks the same random number, they will collide again. After the second collision, each one picks either 0, 1, 2, or 3 at random and waits that number of slot times. If a third collision occurs (the probability of this happening is 0.25), then the next time the number of slots to wait is chosen at random from the interval 0 to $2^3 - 1$.

In general, after i collisions, a random number between 0 and $2^i - 1$ is chosen, and that number of slots is skipped. However, after ten collisions have been reached, the randomization interval is frozen at a maximum of 1023 slots. After 16 collisions, the controller throws in the towel and reports failure back to the computer. Further recovery is up to higher layers.

This algorithm, called **binary exponential backoff**, was chosen to dynamically adapt to the number of stations trying to send. If the randomization interval for all collisions was 1023, the chance of two stations colliding for a second time would be negligible, but the average wait after a collision would be hundreds of slot times, introducing significant delay. On the other hand, if each station always delayed for either zero or one slots, then if 100 stations ever tried to send at once, they would collide over and over until 99 of them picked 0 and the remaining station picked 1, or vice versa. This might take years. By having the randomization interval grow exponentially as more and more consecutive collisions occur, the algorithm ensures a low delay when only a few stations collide but also ensures that the collision is resolved in a reasonable interval when many stations collide.

As described so far, CSMA/CD provides no acknowledgements. Since the mere absence of collisions does not guarantee that bits were not garbled by noise spikes on the cable, for reliable communication the destination must verify the checksum, and if correct, send back an acknowledgement frame to the source. Normally, this acknowledgement would be just another frame as far as the protocol is concerned and would have to fight for channel time just like a data frame. However, a simple modification to the contention algorithm would allow speedy confirmation of frame receipt (Tokoro and Tamaru, 1977). All that would be needed is to reserve the first contention slot following successful transmission for the destination station.

802.3 Performance

Now let us briefly examine the performance of 802.3 under conditions of heavy and constant load, that is, k stations always ready to transmit. A rigorous analysis of the binary exponential backoff algorithm is complicated. Instead we will follow Metcalfe and Boggs (1976) and assume a constant retransmission probability in each slot. If each station transmits during a contention slot with probability p, the probability A that some station acquires the channel in that slot is

$$A = kp(1-p)^{k-1} \tag{4-6}$$

A is maximized when $p = 1/k$, with $A \rightarrow 1/e$ as $k \rightarrow \infty$. The probability that the contention interval has exactly j slots in it is $A(1-A)^{j-1}$, so the mean number of slots per contention is given by

$$\sum_{j=0}^{\infty} jA(1-A)^{j-1} = \frac{1}{A}$$

Since each slot has a duration 2τ, the mean contention interval, w, is $2\tau/A$.

Assuming optimal p, the mean number of contention slots is never more than e, so w is at most $2\tau e \approx 5.4\tau$.

If the mean frame takes P sec to transmit, when many stations have frames to send,

$$\text{Channel efficiency} = \frac{P}{P + 2\tau/A} \tag{4-7}$$

Here we see where the maximum cable distance between any two stations enters into the performance figures, giving rise to topologies other than that of Fig. 4-19(a). The longer the cable, the longer the contention interval. By allowing no more than 2.5 km of cable and four repeaters between any two transceivers, the round-trip time can be bounded to 51.2 μsec, which at 10 Mbps corresponds to 512 bits or 64 bytes, the minimum frame size.

It is instructive to formulate Eq. (4-7) in terms of the frame length, F, the network bandwidth, B, the cable length, L, and the speed of signal propagation, c, for the optimal case of e contention slots per frame. With $P = F/B$, Eq. (4-7) becomes

$$\text{Channel efficiency} = \frac{1}{1 + 2BLe/cF} \tag{4-8}$$

When the second term in the denominator is large, network efficiency will be low. More specifically, increasing network bandwidth or distance (the BL product) reduces efficiency for a given frame size. Unfortunately, much research on network hardware is aimed precisely at increasing this product. People want high bandwidth over long distances (fiber optic MANs, for example), which suggests that 802.3 may not be the best system for these applications.

In Fig. 4-23, the channel efficiency is plotted versus number of ready stations for $2\tau = 51.2$ μsec and a data rate of 10 Mbps using Eq. (4-8). With a 64-byte slot time, it is not surprising that 64-byte frames are not efficient. On the other hand, with 1024-byte frames and an asymptotic value of e 64-byte slots per contention interval, the contention period is 174 bytes long and the efficiency is 0.85.

To determine the mean number of stations ready to transmit under conditions of high load, we can use the following (crude) observation. Each frame ties up the channel for one contention period and one frame transmission time, for a total of $P + w$ sec. The number of frames per second is therefore $1/(P + w)$. If each station generates frames at a mean rate of λ frames/sec, when the system is in state k the total input rate of all unblocked stations combined is $k\lambda$ frames/sec. Since in equilibrium the input and output rates must be identical, we can equate these two expressions and solve for k. (Notice that w is a function of k.) A more sophisticated analysis is given in (Bertsekas and Gallager, 1992).

It is probably worth mentioning that there has been a large amount of theoretical performance analysis of 802.3 (and other networks). Virtually all of this work has assumed that traffic is Poisson. As researchers have begun looking at real

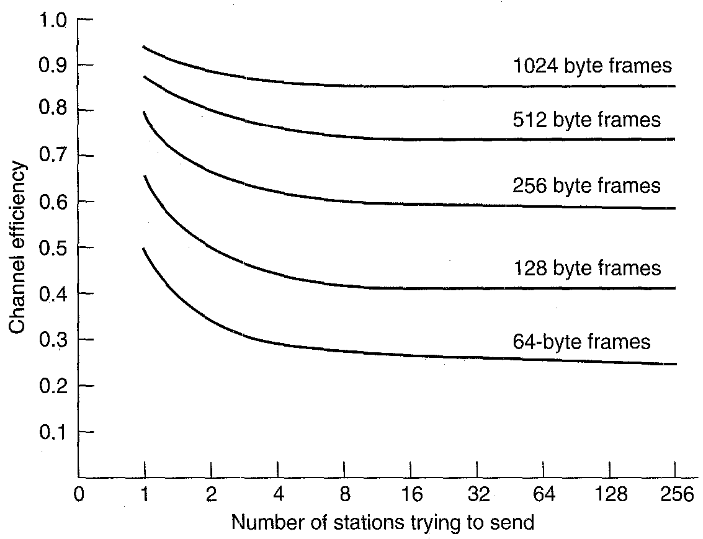


Fig. 4-23. Efficiency of 802.3 at 10 Mbps with 512-bit slot times.

data, it now appears that network traffic is rarely Poisson, but self-similar (Paxson and Floyd, 1994; and Willinger et al., 1995). What this means is that averaging over long periods of time does not smooth out the traffic. The average number of packets in each minute of an hour has as much variance as the average number of packets in each second of a minute. The consequence of this discovery is that most models of network traffic do not apply to the real world and should be taken with a grain (or better yet, a metric ton) of salt.

Switched 802.3 LANs

As more and more stations are added to an 802.3 LAN, the traffic will go up. Eventually, the LAN will saturate. One way out is to go to a higher speed, say from 10 Mbps to 100 Mbps. This solution requires throwing out all the 10 Mbps adaptor cards and buying new ones, which is expensive. If the 802.3 chips are on the computers' main circuit boards, it may not even be possible to replace them.

Fortunately, a different, less drastic solution is possible: a switched 802.3 LAN, as shown in Fig. 4-24. The heart of this system is a switch containing a high-speed backplane and room for typically 4 to 32 plug-in line cards, each containing one to eight connectors. Most often, each connector has a 10Base-T twisted pair connection to a single host computer.

When a station wants to transmit an 802.3 frame, it outputs a standard frame to the switch. The plug-in card getting the frame checks to see if it is destined for one of the other stations connected to the same card. If so, the frame is copied

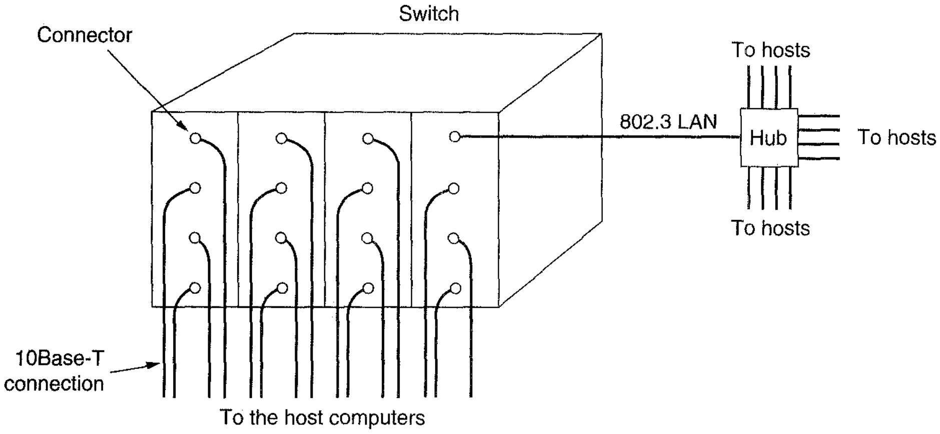


Fig. 4-24. A switched 802.3 LAN.

there. If not, the frame is sent over the high-speed backplane to the destination station's card. The backplane typically runs at over 1 Gbps using a proprietary protocol.

What happens if two machines attached to the same plug-in card transmit frames at the same time? It depends on how the card has been constructed. One possibility is for all the ports on the card to be wired together to form a local on-card LAN. Collisions on this on-card LAN will be detected and handled the same as any other collisions on a CSMA/CD network—with retransmissions using the binary backoff algorithm. With this kind of plug-in card, only one transmission per card is possible at any instant, but all the cards can be transmitting in parallel. With this design, each card forms its own **collision domain**, independent of the others.

With the other kind of plug-in card, each input port is buffered, so incoming frames are stored in the card's on-board RAM as they arrive. This design allows all input ports to receive (and transmit) frames at the same time, for parallel, full-duplex operation. Once a frame has been completely received, the card can then check to see if the frame is destined for another port on the same card, or for a distant port. In the former case it can be transmitted directly to the destination. In the latter case, it must be transmitted over the backplane to the proper card. With this design, each port is a separate collision domain, so collisions do not occur. The total system throughput can often be increased by an order of magnitude over 10Base-5, which has a single collision domain for the entire system.

Since the switch just expects standard 802.3 frames on each input port, it is possible use some of the ports as concentrators. In Fig. 4-24, the port in the upper right-hand corner is connected not to a single station, but to a 12-port hub. As frames arrive at the hub, they contend for the 802.3 LAN in the usual way, including collisions and binary backoff. Successful frames make it to the switch, and are treated there like any other incoming frames: they are switched to the correct output line over the high-speed backplane. If all the input ports are connected to hubs, rather than to individual stations, the switch just becomes an 802.3 to 802.3 bridge. We will study bridges later in this chapter.

4.3.2. IEEE Standard 802.4: Token Bus

Although 802.3 is widely used in offices, during the development of the 802 standard, people from General Motors and other companies interested in factory automation had serious reservations about it. For one thing, due to the probabilistic MAC protocol, with a little bad luck a station might have to wait arbitrarily long to send a frame (i.e., the worst case is unbounded). For another, 802.3 frames do not have priorities, making them unsuited for real-time systems in which important frames should not be held up waiting for unimportant frames.

A simple system with a known worst case is a ring in which the stations take turns sending frames. If there are n stations and it takes T sec to send a frame, no frame will ever have to wait more than nT sec to be sent. The factory automation people in the 802 committee liked the conceptual idea of a ring but did not like the physical implementation because a break in the ring cable would bring the whole network down. Furthermore, they noted that a ring is a poor fit to the linear topology of most assembly lines. As a result, a new standard was developed, having the robustness of the 802.3 broadcast cable, but the known worst-case behavior of a ring.

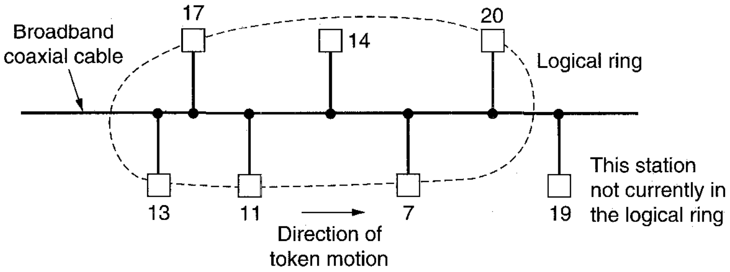

Broadband coaxial cable
17
14
20
Logical ring
This station not currently in the logical ring
13
11
7
19
Direction of token motion

Fig. 4-25. A token bus.

This standard, 802.4 (Dirvin and Miller, 1986; and IEEE, 1985b), describes a LAN called a **token bus**. Physically, the token bus is a linear or tree-shaped cable onto which the stations are attached. Logically, the stations are organized into a ring (see Fig. 4-25), with each station knowing the address of the station to its "left" and "right." When the logical ring is initialized, the highest numbered station may send the first frame. After it is done, it passes permission to its immediate neighbor by sending the neighbor a special control frame called a **token**. The token propagates around the logical ring, with only the token holder being permitted to transmit frames. Since only one station at a time holds the token, collisions do not occur.

An important point to realize is that the physical order in which the stations are connected to the cable is not important. Since the cable is inherently a broadcast medium, each station receives each frame, discarding those not addressed to it. When a station passes the token, it sends a token frame specifically addressed to its logical neighbor in the ring, irrespective of where that station is physically located on the cable. It is also worth noting that when stations are first powered on, they will not be in the ring (e.g., stations 14 and 19 in Fig. 4-25), so the MAC protocol has provisions for adding stations to, and deleting stations from, the ring.

The 802.4 MAC protocol is very complex, with each station having to maintain ten different timers and more than two dozen internal state variables. The 802.4 standard is much longer than 802.3, filling more than 200 pages. The two standards are also quite different in style, with 802.3 giving the protocols as Pascal procedures, whereas 802.4 gives them as finite state machines, with the actions written in Ada®.

For the physical layer, the token bus uses the 75-ohm broadband coaxial cable used for cable television. Both single- and dual-cable systems are allowed, with or without head-ends. Three different analog modulation schemes are permitted: phase continuous frequency shift keying, phase coherent frequency shift keying, and multilevel duobinary amplitude modulated phase shift keying. Speeds of 1, 5, and 10 Mbps are possible. Furthermore, the modulation schemes not only provide ways to represent 0, 1, and idle on the cable, but also three other symbols used for network control. All in all, the physical layer is totally incompatible with 802.3, and a lot more complicated.

The Token Bus MAC Sublayer Protocol

When the ring is initialized, stations are inserted into it in order of station address, from highest to lowest. Token passing is also done from high to low addresses. Each time a station acquires the token, it can transmit frames for a certain amount of time; then it must pass the token on. If the frames are short enough, several consecutive frames may be sent. If a station has no data, it passes the token immediately upon receiving it.

The token bus defines four priority classes, 0, 2, 4, and 6 for traffic, with 0 the lowest and 6 the highest. It is easiest to think of each station internally being divided into four substations, one at each priority level. As input comes in to the MAC sublayer from above, the data are checked for priority and routed to one of the four substations. Thus each substation maintains its own queue of frames to be transmitted.

When the token comes into the station over the cable, it is passed internally to the priority 6 substation, which may begin transmitting frames, if it has any.

When it is done (or when its timer expires), the token is passed internally to the priority 4 substation, which may then transmit frames until its timer expires, at which point the token is passed internally to the priority 2 substation. This process is repeated until either the priority 0 substation has sent all its frames or its timer has expired. Either way, at this point the token is sent to the next station in the ring.

Without getting into all the details of how the various timers are managed, it should be clear that by setting the timers properly, we can ensure that a guaranteed fraction of the total token-holding time can be allocated to priority 6 traffic. The lower priorities will have to live with what is left over. If the higher priority substations do not need all of their allocated time, the lower priority substations can have the unused portion, so it is not wasted.

This priority scheme, which guarantees priority 6 traffic a known fraction of the network bandwidth, can be used to implement real-time traffic. For example, suppose the parameters of a 50-station network running at 10 Mbps have been adjusted to give priority 6 traffic 1/3 of the bandwidth. Then each station has a guaranteed 67 kbps for priority 6 traffic. This bandwidth could be used to synchronize robots on an assembly line or carry one digital voice channel per station, with a little left over for control information.

The token bus frame format is shown in Fig. 4-26. It is unfortunately different from the 802.3 frame format. The preamble is used to synchronize the receiver's clock, as in 802.3, except that here it may be as short as 1 byte. The *Starting delimiter* and *Ending delimiter* fields are used to mark the frame boundaries. Both of these fields contain analog encoding of symbols other than 0s and 1s, so that they cannot occur accidentally in the user data. As a result, no length field is needed.

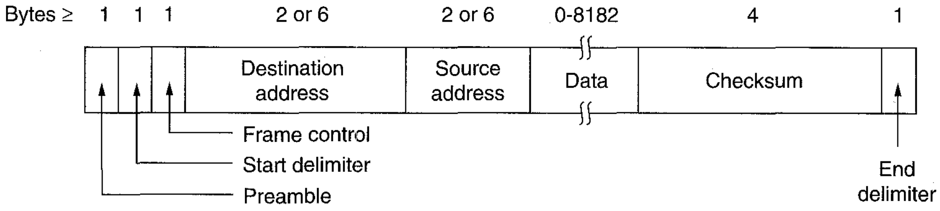


Fig. 4-26. The 802.4 frame format.

The *Frame control* field is used to distinguish data frames from control frames. For data frames, it carries the frame's priority. It can also carry an indicator requiring the destination station to acknowledge correct or incorrect receipt of the frame. Without this indicator, the destination would not be allowed to send anything because it does not have the token. This indicator turns the token bus into something resembling the acknowledgement scheme of Tokoro and Tamaru.

For control frames, the *Frame control* field is used to specify the frame type.

The allowed types include token passing and various ring maintenance frames, including the mechanism for letting new stations enter the ring, the mechanism for allowing stations to leave the ring, and so on. Note that the 802.3 protocol does not have any control frames. All the MAC layer does there is provide a way to get frames onto the cable; it does not care what is in them.

The *Destination address* and *Source address* fields are the same as in 802.3 (yes, the two groups did talk to each other; no, they did not agree on very much). As in 802.3, a given network must use all 2-byte addresses or all 6-byte addresses, not a mixture on the same cable. The initial 802.4 standard allows either size. The individual and group addressing and the local and global address assignments are identical to 802.3.

The *Data* field may be up to 8182 bytes long when 2-byte addresses are used, and up to 8174 bytes long when 6-byte addresses are used. This is more than five times as long as the maximum 802.3 frame, which was made short to prevent one station from hogging the channel too long. With the token bus, the timers can be used as an antihogging measure, but it is nice to be able to send long frames when real-time traffic is not an issue. The *Checksum* is used to detect transmission errors. It uses the same algorithm and polynomial as 802.3.

The token bus control frames are shown in Fig. 4-27. They will be discussed below. The only one we have seen so far is the *token* frame, used to pass the token from station to station. Most of the rest relate to adding and deleting stations from the logical ring.

Frame control field	Name	Meaning
00000000	Claim_token	Claim token during ring initialization
00000001	Solicit_successor_1	Allow stations to enter the ring
00000010	Solicit_successor_2	Allow stations to enter the ring
00000011	Who_follows	Recover from lost token
00000100	Resolve_contention	Used when multiple stations want to enter
00001000	Token	Pass the token
00001100	Set_successor	Allow station to leave the ring

Fig. 4-27. The token bus control frames.

Logical Ring Maintenance

From time to time, stations are powered on and want to join the ring. Other are turned off and want to leave. The MAC sublayer protocol provides a detailed specification of exactly how this is done while maintaining the known worst case bound on token rotation. Below we will just briefly sketch the mechanisms used.

Once the ring has been established, each station's interface maintains the addresses of the predecessor and successor stations internally. Periodically, the token holder sends one of the SOLICIT_SUCCESSOR frames shown in Fig. 4-27 to solicit bids from stations that wish to join the ring. The frame gives the sender's address and the successor's address. Stations inside that range may bid to enter (to keep the ring sorted in descending order of station address).

If no station bids to enter within a slot time (2τ, as in 802.3), the **response window** is closed and the token holder continues with its normal business. If exactly one station bids to enter, it is inserted into the ring and becomes the token holder's successor.

If two or more stations bid to enter, their frames will collide and be garbled, as in 802.3. The token holder then runs an arbitration algorithm, starting with the broadcast of a RESOLVE_CONTENTION frame. The algorithm is a variation of binary countdown, using two bits at a time.

Furthermore, all station interfaces maintain two random bits inside. These bits are used to delay all bids by 0, 1, 2, or 3 slot times, to further reduce contention. In other words, two stations only collide on a bid if the current two address bits being used are the same and they happen to have the same two random bits. To prevent stations that must wait 3 slot times from being at a permanent disadvantage, the random bits are regenerated every time they are used or periodically every 50 msec.

The solicitation of new stations may not interfere with the guaranteed worst case for token rotation. Each station has a timer that is reset whenever it acquires the token. When the token comes in, the old value of this timer (i.e., the previous token rotation time) is inspected just before the timer is reset. If it exceeds a certain threshold value, there has been too much traffic recently, so no bids may be solicited this time around. In any event, only one station may enter at each solicitation, to put a bound on how much time can be consumed in ring maintenance. No guarantee is provided for how long a station may have to wait to join the ring when traffic is heavy, but in practice it should not be more than a few seconds. This uncertainty is unfortunate, making 802.4 less suitable for real-time systems than its supporters often claim.

Leaving the ring is easy. A station, *X*, with successor *S*, and predecessor *P*, leaves the ring, by sending *P* a SET_SUCCESSOR frame telling it that henceforth its successor is *S* instead of *X*. Then *X* just stops transmitting.

Ring initialization is a special case of adding new stations. Consider an idle system with all stations powered off. When the first station comes on-line, it notices that there is no traffic for a certain period. Then it sends a CLAIM_TOKEN frame. Not hearing any competitors contending for the token, it creates a token and sets up a ring containing only itself. Periodically, it solicits bids for new stations to join. As new stations are powered on, they will respond to these bids and join the ring using the contention algorithm described above. Eventually, every station that wants to join the ring will be able to do so. If the first two stations are powered on simultaneously, the protocol deals with this by letting them bid for the token using the standard modified binary countdown algorithm and the two random bits.

Due to transmission errors or hardware failures, problems can arise with the logical ring or the token. For example, if a station tries to pass the token to a station that has gone down, what happens? The solution is straightforward. After passing the token, a station listens to see if its successor either transmits a frame or passes the token. If it does neither, the token is passed a second time.

If that also fails, the station transmits a WHO_FOLLOWS frame specifying the address of its successor. When the failed station's successor sees a WHO_FOLLOWS frame naming its predecessor, it responds by sending a SET_SUCCESSOR frame to the station whose successor failed, naming itself as the new successor. In this way, the failed station is removed from the ring.

Now suppose that a station fails to pass the token to its successor and also fails to locate the successor's successor, which may also be down. It adopts a new strategy by sending a SOLICIT_SUCCESSOR_2 frame to see if *anyone* else is still alive. Once again the standard contention protocol is run, with all stations that want to be in the ring now bidding for a place. Eventually, the ring is re-established.

Another kind of problem occurs if the token holder goes down and takes the token with it. This problem is solved using the ring initialization algorithm. Each station has a timer that is reset whenever a frame appears on the network. When this timer hits a threshold value, the station issues a CLAIM_TOKEN frame, and the modified binary countdown algorithm with random bits determines who gets the token.

Still another problem is multiple tokens. If a station holding the token notices a transmission from another station, it discards its token. If there were two, there would now be one. If there were more than two, this process would be repeated sooner or later until all but one were discarded. If, by accident, all the tokens are discarded, then the lack of activity will cause one or more stations to try to claim the token.

4.3.3. IEEE Standard 802.5: Token Ring

Ring networks have been around for many years (Pierce, 1972) and have long been used for both local and wide area networks. Among their many attractive features is the fact that a ring is not really a broadcast medium, but a collection of individual point-to-point links that happen to form a circle. Point-to-point links involve a well-understood and field-proven technology and can run on twisted pair, coaxial cable, or fiber optics. Ring engineering is also almost entirely digital, whereas 802.3, for example, has a substantial analog component for collision detection. A ring is also fair and has a known upper bound on channel access.

For these reasons, IBM chose the ring as its LAN and IEEE has included the **token ring** standard as 802.5 (IEEE, 1985c; Latif et al., 1992).

A major issue in the design and analysis of any ring network is the "physical length" of a bit. If the data rate of the ring is R Mbps, a bit is emitted every $1/R$ μsec. With a typical signal propagation speed of about 200 m/μsec, each bit occupies $200/R$ meters on the ring. This means, for example, that a 1-Mbps ring whose circumference is 1000 meters can contain only 5 bits on it at once. The implications of the number of bits on the ring will become clearer later.

As mentioned above, a ring really consists of a collection of ring interfaces connected by point-to-point lines. Each bit arriving at an interface is copied into a 1-bit buffer and then copied out onto the ring again. While in the buffer, the bit can be inspected and possibly modified before being written out. This copying step introduces a 1-bit delay at each interface. A ring and its interfaces are shown in Fig. 4-28.

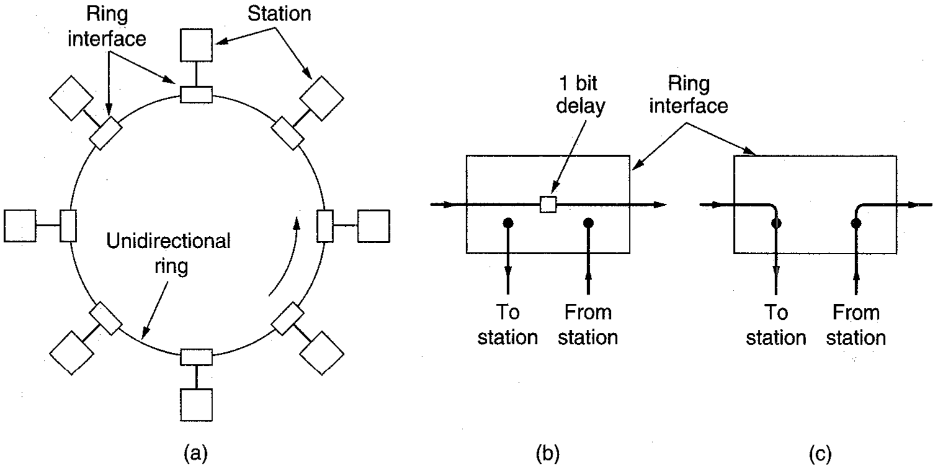


Fig. 4-28. (a) A ring network. (b) Listen mode. (c) Transmit mode.

In a token ring a special bit pattern, called the **token**, circulates around the ring whenever all stations are idle. When a station wants to transmit a frame, it is required to seize the token and remove it from the ring before transmitting. This action is done by inverting a single bit in the 3-byte token, which instantly changes it into the first 3 bytes of a normal data frame. Because there is only one token, only one station can transmit at a given instant, thus solving the channel access problem the same way the token bus solves it.

An implication of the token ring design is that the ring itself must have a sufficient delay to contain a complete token to circulate when all stations are idle. The delay has two components: the 1-bit delay introduced by each station, and the signal propagation delay. In almost all rings, the designers must assume that stations may be powered down at various times, especially at night. If the interfaces are powered from the ring, shutting down the station has no effect on the interface, but if the interfaces are powered externally, they must be designed to connect the input to the output when power goes down, thus removing the 1-bit delay. The point here is that on a short ring an artificial delay may have to be inserted into the ring at night to ensure that a token can be contained on it.

Ring interfaces have two operating modes, listen and transmit. In listen mode, the input bits are simply copied to output, with a delay of 1 bit time, as shown in Fig. 4-28(b). In transmit mode, which is entered only after the token has been seized, the interface breaks the connection between input and output, entering its own data onto the ring. To be able to switch from listen to transmit mode in 1 bit time, the interface usually needs to buffer one or more frames itself rather than having to fetch them from the station on such short notice.

As bits that have propagated around the ring come back, they are removed from the ring by the sender. The sending station can either save them, to compare with the original data to monitor ring reliability, or discard them. Because the entire frame never appears on the ring at one instant, this ring architecture puts no limit on the size of the frames. After a station has finished transmitting the last bit of its last frame, it must regenerate the token. When the last bit of the frame has gone around and come back, it must be removed, and the interface must switch back into listen mode immediately, to avoid removing the token that might follow if no other station has removed it.

It is straightforward to handle acknowledgements on a token ring. The frame format need only include a 1-bit field for acknowledgements, initially zero. When the destination station has received a frame, it sets the bit. Of course, if the acknowledgement means that the checksum has been verified, the bit must follow the checksum, and the ring interface must be able to verify the checksum as soon as its last bit has arrived. When a frame is broadcast to multiple stations, a more complicated acknowledgement mechanism must be used (if any is used at all).

When traffic is light, the token will spend most of its time idly circulating around the ring. Occasionally a station will seize it, transmit a frame, and then output a new token. However, when the traffic is heavy, so that there is a queue at each station, as soon as a station finishes its transmission and regenerates the token, the next station downstream will see and remove the token. In this manner the permission to send rotates smoothly around the ring, in round-robin fashion. The network efficiency can begin to approach 100 percent under conditions of heavy load.

Now let us turn from token rings in general to the 802.5 standard in particular. At the physical layer, 802.5 calls for shielded twisted pairs running at 1 or 4 Mbps, although IBM later introduced a 16-Mbps version. Signals are encoded using differential Manchester encoding [see Fig. 4-20(c)] with high and low being positive and negative signals of absolute magnitude 3.0 to 4.5 volts. Normally, differential Manchester encoding uses high-low or low-high for each bit, but

802.5 also uses high-high and low-low in certain control bytes (e.g., to mark the start and end of a frame). These nondata signals always occur in consecutive pairs so as not to introduce a DC component into the ring voltage.

One problem with a ring network is that if the cable breaks somewhere, the ring dies. This problem can be solved very elegantly by the use of a **wire center**, as shown in Fig. 4-29. While logically still a ring, physically each station is connected to the wire center by a cable containing (at least) two twisted pairs, one for data to the station and one for data from the station.

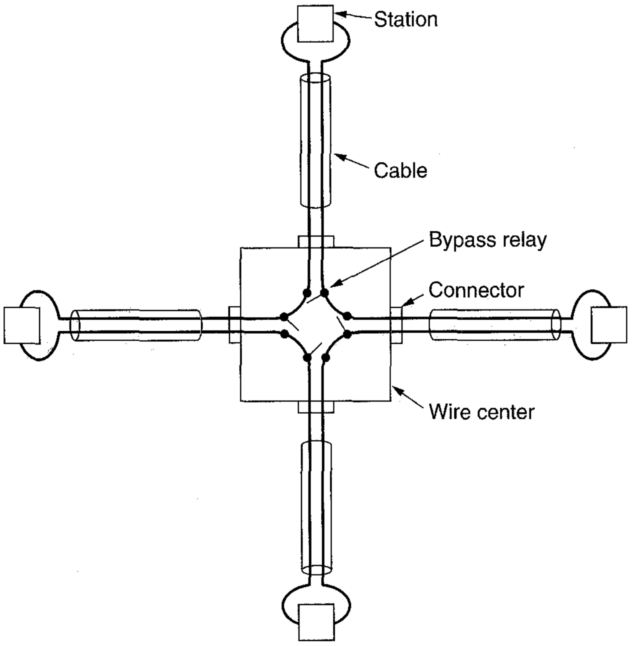


Fig. 4-29. Four stations connected via a wire center.

Inside the wire center are bypass relays that are energized by current from the stations. If the ring breaks or a station goes down, loss of the drive current will release the relay and bypass the station. The relays can also be operated by software to permit diagnostic programs to remove stations one at a time to find faulty stations and ring segments. The ring can then continue operation with the bad segment bypassed. Although the 802.5 standard does not formally require this kind of ring, often called a **star-shaped ring** (Saltzer et al., 1983), most 802.5 LANs, in fact, do use wire centers to improve their reliability and maintainability.

When a network consists of many clusters of stations far apart, a topology with multiple wire centers can be used. Just imagine that the cable to one of the stations in Fig. 4-29 were replaced by a cable to a distant wire center. Although logically all the stations are on the same ring, the wiring requirements are greatly reduced. An 802.5 ring using a wire center has a similar topology to an 802.3 10Base-T hub-based network, but the formats and protocols are different.

The Token Ring MAC Sublayer Protocol

The basic operation of the MAC protocol is straightforward. When there is no traffic on the ring, a 3-byte token circulates endlessly, waiting for a station to seize it by setting a specific 0 bit to a 1 bit, thus converting the token into the start-of-frame sequence. The station then outputs the rest of a normal data frame, as shown in Fig. 4-30.

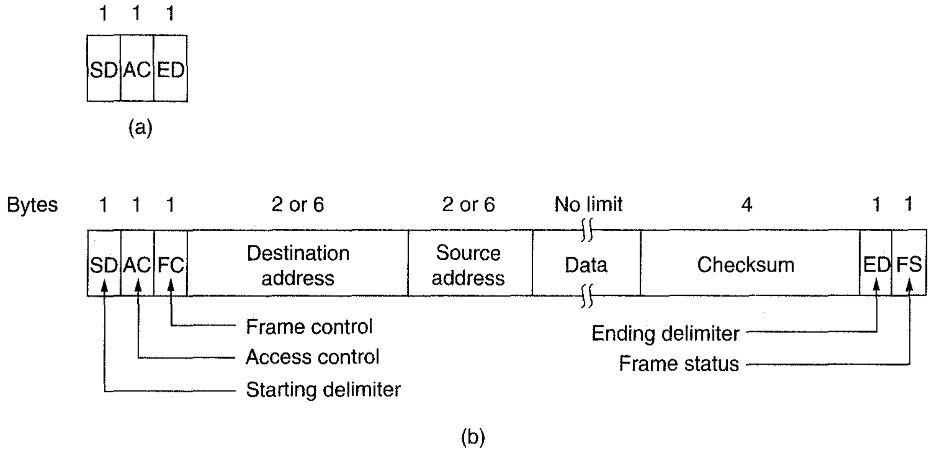


Fig. 4-30. (a) Token format. (b) Data frame format.

Under normal conditions, the first bit of the frame will go around the ring and return to the sender before the full frame has been transmitted. Only a very long ring will be able to hold even a short frame. Consequently, the transmitting station must drain the ring while it continues to transmit. As shown in Fig. 4-28(c), this means that the bits that have completed the trip around the ring come back to the sender and are removed.

A station may hold the token for the **token-holding time**, which is 10 msec unless an installation sets a different value. If there is enough time left after the first frame has been transmitted to send more frames, these may be sent as well. After all pending frames have been transmitted or the transmission of another frame would exceed the token-holding time, the station regenerates the 3-byte token frame and puts it out onto the ring.

The *Starting delimiter* and *Ending delimiter* fields of Fig. 4-30(b) mark the beginning and ending of the frame. Each contains invalid differential Manchester patterns (HH and LL) to distinguish them from data bytes. The *Access control* byte contains the token bit, and also the *Monitor bit*, *Priority bits*, and *Reservation bits* (described below). The *Frame control* byte distinguishes data frames from various possible control frames.

Next come the *Destination address* and *Source address* fields, which are the same as in 802.3 and 802.4. These are followed by the data, which may be as long as necessary, provided that the frame can still be transmitted within the token-holding time. The *Checksum* field, like the destination and source addresses, is also the same as 802.3 and 802.4.

An interesting byte not present in the other two protocols is the *Frame status* byte. It contains the *A* and *C* bits. When a frame arrives at the interface of a station with the destination address, the interface turns on the *A* bit as it passes through. If the interface copies the frame to the station, it also turns on the *C* bit. A station might fail to copy a frame due to lack of buffer space or other reasons.

When the sending station drains the frame from the ring, it examines the *A* and *C* bits. Three combinations are possible:

1. $A = 0$ and $C = 0$: destination not present or not powered up.
2. $A = 1$ and $C = 0$: destination present but frame not accepted.
3. $A = 1$ and $C = 1$: destination present and frame copied.

This arrangement provides an automatic acknowledgement for each frame. If a frame is rejected but the station is present, the sender has the option of trying again in a little while. The *A* and *C* bits are present twice in the *Frame status* to increase reliability inasmuch as they are not covered by the checksum.

The *Ending delimiter* contains an *E* bit which is set if any interface detects an error (e.g., a non-Manchester pattern where that is not permitted). It also contains a bit that can be used to mark the last frame in a logical sequence, sort of like an end-of-file bit.

The 802.5 protocol has an elaborate scheme for handling multiple priority frames. The 3-byte token frame contains a field in the middle byte giving the priority of the token. When a station wants to transmit a priority *n* frame, it must wait until it can capture a token whose priority is less than or equal to *n*. Furthermore, when a data frame goes by, a station can try to reserve the next token by writing the priority of the frame it wants to send into the frame's *Reservation bits*. However, if a higher priority has already been reserved there, the station may not make a reservation. When the current frame is finished, the next token is generated at the priority that has been reserved.

A little thought will show that this mechanism acts like a ratchet, always jacking the reservation priority higher and higher. To eliminate this problem, the protocol contains some complex rules. The essence of the idea is that the station raising the priority is responsible for lowering the priority again when it is done.

Notice that this priority scheme is substantially different from the token bus scheme, in which each station always gets its fair share of the bandwidth, no matter what other stations are doing. In the token ring, a station with only low priority frames may starve to death waiting for a low priority token to appear. Clearly, the two committees had different taste when trading off good service for high priority traffic versus fairness to all stations.

Ring Maintenance

The token bus protocol goes to considerable lengths to do ring maintenance in a fully decentralized way. The token ring protocol handles maintenance quite differently. Each token ring has a **monitor station** that oversees the ring. If the monitor goes down, a contention protocol ensures that another station is quickly elected as monitor. (Every station has the capability of becoming the monitor.) While the monitor is functioning properly, it alone is responsible for seeing that the ring operates correctly.

When the ring comes up or any station notices that there is no monitor, it can transmit a CLAIM TOKEN control frame. If this frame circumnavigates the ring before any other CLAIM TOKEN frames are sent, the sender becomes the new monitor (each station has monitor capability built in). The token ring control frames are shown in Fig. 4-31.

Control field	Name	Meaning
00000000	Duplicate address test	Test if two stations have the same address
00000010	Beacon	Used to locate breaks in the ring
00000011	Claim token	Attempt to become monitor
00000100	Purge	Reinitialize the ring
00000101	Active monitor present	Issued periodically by the monitor
00000110	Standby monitor present	Announces the presence of potential monitors

Fig. 4-31. Token ring control frames.

Among the monitor's responsibilities are seeing that the token is not lost, taking action when the ring breaks, cleaning the ring up when garbled frames appear, and watching out for orphan frames. An orphan frame occurs when a station transmits a short frame in its entirety onto a long ring and then crashes or is powered down before the frame can be drained. If nothing is done, the frame will circulate forever.

To check for lost tokens, the monitor has a timer that is set to the longest possible tokenless interval: each station transmitting for the full token-holding time. If this timer goes off, the monitor drains the ring and issues a new token.

When a garbled frame appears, the monitor can detect it by its invalid format or checksum, open the ring to drain it, and issue a new token when the ring has been cleaned up. Finally, the monitor detects orphan frames by setting the *monitor* bit in the *Access control* byte whenever it passes through. If an incoming frame has this bit set, something is wrong since the same frame has passed the monitor twice without having been drained, so the monitor drains it.

One last monitor function concerns the length of the ring. The token is 24 bits long, which means that the ring must be big enough to hold 24 bits. If the 1-bit delays in the stations plus the cable length add up to less than 24 bits, the monitor inserts extra delay bits so that a token can circulate.

One maintenance function that cannot be handled by the monitor is locating breaks in the ring. When a station notices that either of its neighbors appears to be dead, it transmits a BEACON frame giving the address of the presumably dead station. When the beacon has propagated around as far as it can, it is then possible to see how many stations are down and delete them from the ring using the bypass relays in the wire center, all without human intervention.

It is instructive to compare the approaches taken to controlling the token bus and the token ring. The 802.4 committee was scared to death of having any centralized component that could fail in some unexpected way and take the system down with it. Therefore they designed a system in which the current token holder had special powers (e.g., soliciting bids to join the ring), but no station was otherwise different from the others (e.g., currently assigned administrative responsibility for maintenance).

The 802.5 committee, on the other hand, felt that having a centralized monitor made handling lost tokens, orphan frames and so on much easier. Furthermore, in a normal system, stations hardly ever crash, so occasionally having to put up with contention for a new monitor is not a great hardship. The price paid is that if the monitor ever really goes berserk but continues to issue ACTIVE MONITOR PRESENT control frames periodically, no station will ever challenge it. Monitors cannot be impeached.

This difference in approach comes from the different application areas the two committees had in mind. The 802.4 committee was thinking in terms of factories with large masses of metal moving around under computer control. Network failures could result in severe damage and had to be prevented at all costs. The 802.5 committee was interested in office automation, where a failure once in a rare while could be tolerated as the price for a simpler system. Whether 802.4 is, in fact, more reliable than 802.5 is a matter of some controversy.

4.3.4. Comparison of 802.3, 802.4, and 802.5

With three different and incompatible LANs available, each with different properties, many organizations are faced with the question: Which one should we install? In this section we will look at all three of the 802 LAN standards, pointing out their strengths and weaknesses, comparing and contrasting them.

To start with, it is worth noting that the three LAN standards use roughly similar technology and get roughly similar performance. While computer scientists and engineers can discuss the merits of coax versus twisted pair for hours on end if given half a chance, the people in the marketing, personnel, or accounting departments probably do not really care that much one way or the other.

Let us start with the advantages of 802.3. It is far and away the most widely used type at present, with a huge installed base and considerable operational experience. The protocol is simple. Stations can be installed on the fly, without taking the network down. A passive cable is used and modems are not required. Furthermore, the delay at low load is practically zero (stations do not have to wait for a token; they just transmit immediately).

On the other hand, 802.3 has a substantial analog component. Each station has to be able to detect the signal of the weakest other station, even when it itself is transmitting, and all of the collision detect circuitry in the transceiver is analog. Due to the possibility of having frames aborted by collisions, the minimum valid frame is 64 bytes, which represents substantial overhead when the data consist of just a single character from a terminal.

Furthermore, 802.3 is nondeterministic, which is often inappropriate for real-time work [although some real-time work is possible by simulating a token ring in software (Venkatramani and Chiueh, 1995)]. It also has no priorities. The cable length is limited to 2.5 km (at 10 Mbps) because the round-trip cable length determines the slot time, hence the performance. As the speed increases, the efficiency drops because the frame transmission times drop but the contention interval does not (the slot width is 2τ no matter what the data rate is). Alternatively, the cable has to be made shorter. Also, at high load, the presence of collisions becomes a major problem and can seriously affect the throughput.

Now let us consider 802.4, the token bus. It uses highly reliable cable television equipment, which is available off-the-shelf from numerous vendors. It is more deterministic than 802.3, although repeated losses of the token at critical moments can introduce more uncertainty than its supporters like to admit. It can handle short minimum frames.

Token bus also supports priorities and can be configured to provide a guaranteed fraction of the bandwidth to high-priority traffic, such as digitized voice. It also has excellent throughput and efficiency at high load, effectively becoming TDM. Finally, broadband cable can support multiple channels, not only for data, but also for voice and television.

On the down side, broadband systems use a lot of analog engineering and include modems and wideband amplifiers. The protocol is extremely complex and has substantial delay at low load (stations must always wait for the token, even in an otherwise idle system). Finally, it is poorly suited for fiber optic implementations and has a small installed base of users.

Now consider the token ring. It uses point-to-point connections, meaning that the engineering is easy and can be fully digital. Rings can be built using virtually any transmission medium from carrier pigeon to fiber optics. The standard twisted pair is cheap and simple to install. The use of wire centers make the token ring the only LAN that can detect and eliminate cable failures automatically.

Like the token bus, priorities are possible, although the scheme is not as fair. Also like the token bus, short frames are possible, but unlike the token bus, so are arbitrarily large ones, limited only by the token-holding time. Finally, the throughput and efficiency at high load are excellent, like the token bus and unlike 802.3.

The major minus is the presence of a centralized monitor function, which introduces a critical component. Even though a dead monitor can be replaced, a sick one can cause headaches. Furthermore, like all token passing schemes, there is always delay at low load because the sender must wait for the token.

It is also worth pointing out that there have been numerous studies of all three LANs. The principal conclusion we can draw from these studies is that we can draw no conclusions from them. One can always find a set of parameters that makes one of the LANs look better than the others. Under most circumstances, all three perform well, so that factors other than the performance are probably more important when making a choice.

4.3.5. IEEE Standard 802.6: Distributed Queue Dual Bus

None of the 802 LANs we have studied so far are suitable for use as a MAN. Cable length limitations and performance problems when thousands of stations are connected limits them to campus-sized areas. For networks covering an entire city, IEEE defined one MAN, called **DQDB** (**Distributed Queue Dual Bus**), as standard 802.6. In this section we will examine how it works. For additional information, see (Kessler and Train, 1992). A bibliography listing 171 papers about DQDB is given in (Sadiku and Arvind, 1994).

The basic geometry of 802.6 is illustrated in Fig. 1-4. Two parallel, unidirectional buses snake through the city, with stations attached to both buses in parallel. Each bus has a head-end, which generates a steady stream of 53-byte cells. Each cell travels downstream from the head-end. When it reaches the end, it falls off the bus.

Each cell carries a 44-byte payload field, making it compatible with some AAL modes. Each cell also holds two protocol bits, *Busy*, set to indicate that a cell is occupied, and *Request*, which can be set when a station wants to make a request.

To transmit a cell, a station has to know whether the destination is to the left of it or to the right of it. If the destination is to the right, the sender uses bus *A*. Otherwise, it uses bus *B*. Data are inserted onto either bus using a wired-OR circuit, so failure of a station does not take down the network.

Unlike all the other 802 LAN protocols, 802.6 is not greedy. In all the others, if a station gets the chance to send, it will. Here, stations queue up in the order they became ready to send and transmit in FIFO order. The interesting part about the protocol is how it achieves FIFO order without having a central queue.

The basic rule is that stations are polite: they defer to stations downstream from them. This politeness is needed to prevent a situation in which the station nearest the head-end simply grabs all the empty cells as they come by and fills them up, starving everyone downstream. For simplicity, we will only examine transmission on bus *A*, but the same story holds for bus *B* as well.

To simulate the FIFO queue, each station maintains two counters, *RC* and *CD*. *RC* (*Request Counter*) counts the number of downstream requests pending until the station itself has a frame to send. At that point, *RC* is copied to *CD*, *RC* is reset to 0, and now counts the number of requests made after the station became ready. For example, if $CD = 3$ and $RC = 2$ for station *k*, the next three empty cells that pass by station *k* are reserved for downstream stations, then station *k* may send, then two more cells are reserved for downstream stations. For simplicity, we assume a station can have only one cell ready for transmission at a time.

To send a cell, a station must first make a reservation by setting the *Request* bit in some cell on the reverse bus (i.e., on bus *B* for a transmission that will later take place on bus *A*). As this cell propagates down the reverse bus, every station along the way notes it and increments its *RC*. To illustrate this concept, we will use an example. Initially, all the *RC* counters are 0, and no cells are queued up, as shown in Fig. 4-32(a). Then station *D* makes a request, which causes stations, *C*, *B*, and *A*, to increment their *RC* counters, as shown in Fig. 4-32(b). After that, *B* makes a request, copying its current *RC* value into *CD*, leading to the situation of Fig. 4-32(c).

At this point, the head-end on bus *A* generates an empty cell. As it passes by *B*, that station sees that its $CD > 0$, so it may not use the empty cell. (When a station has a cell queued, *CD* represents its position in the queue, with 0 being the front of the queue.) Instead it decrements *CD*. When the still-empty cell gets to *D*, that station sees that $CD = 0$, meaning that no one is ahead of it on the queue, so it ORs its data into the cell and sets the *Busy* bit. After the transmissions are done, we have the situation of Fig. 4-32(d).

When the next empty cell is generated, station *B* sees that it is now at the head of the queue, and seizes the cell (by setting 1 bit), as illustrated in Fig. 4-32(e). In this way, stations queue up to take turns, without a centralized queue manager.

DQDB systems are now being installed by many carriers throughout entire cities. Typically they run for up to 160 km at speeds of 44.736 Mbps (T3).

4.3.6. IEEE Standard 802.2: Logical Link Control

It is now perhaps time to step back and compare what we have learned in this chapter with what we studied in the previous one. In Chap. 3, we saw how two machines could communicate reliably over an unreliable line by using various

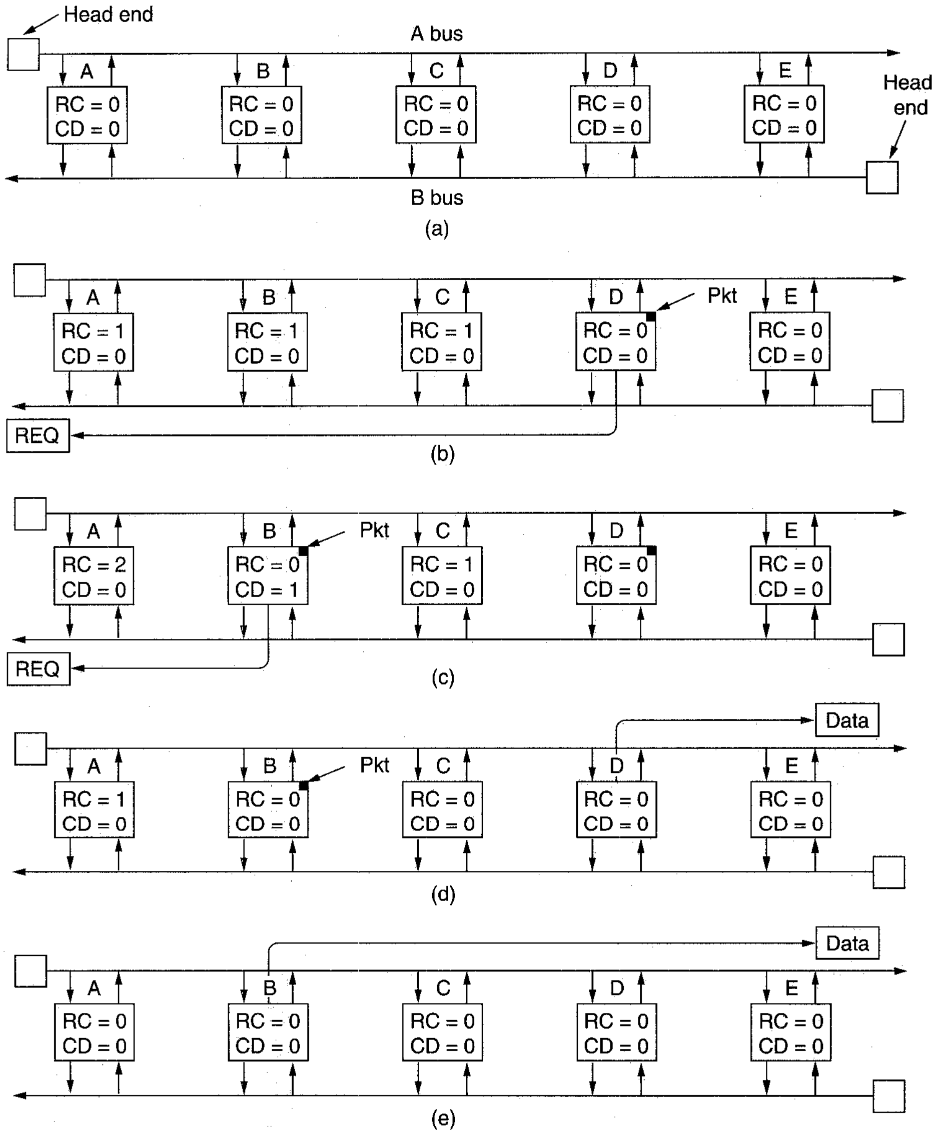


Fig. 4-32. (a) Initially the MAN is idle. (b) After *D* makes a request. (c) After *B* makes a request. (d) After *D* transmits. (e) After *B* transmits.

data link protocols. These protocols provided error control (using acknowledgements) and flow control (using a sliding window).

In contrast, in this chapter, we have not said a word about reliable communication. All that the 802 LANs and MAN offer is a best-efforts datagram service.

Sometimes, this service is adequate. For example, for transporting IP packets, no guarantees are required or even expected. An IP packet can just be inserted into an 802 payload field and sent on its way. If it gets lost, so be it.

Nevertheless, there are also systems in which an error-controlled, flow-controlled data link protocol is desired. IEEE has defined one that can run on top of all the 802 LAN and MAN protocols. In addition, this protocol, called **LLC** (**Logical Link Control**), hides the differences between the various kinds of 802 networks by providing a single format and interface to the network layer. This format, interface, and protocol are all closely based on OSI. LLC forms the upper half of the data link layer, with the MAC sublayer below it, as shown in Fig. 4-33.

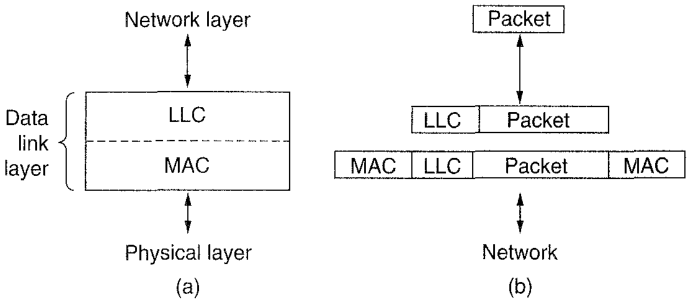


Fig. 4-33. (a) Position of LLC. (b) Protocol formats.

Typical usage of LLC is as follows. The network layer on the sending machine passes a packet to LLC using the LLC access primitives. The LLC sublayer then adds an LLC header, containing sequence and acknowledgement numbers. The resulting structure is then inserted into the payload field of an 802.x frame and transmitted. At the receiver, the reverse process takes place.

LLC provides three service options: unreliable datagram service, acknowledged datagram service, and reliable connection-oriented service. The LLC header is based on the older HDLC protocol. A variety of different formats are used for data and control. For acknowledged datagram or connection-oriented service, the data frames contain a source address, a destination address, a sequence number, an acknowledgement number, and a few miscellaneous bits. For unreliable datagram service, the sequence number and acknowledgement number are omitted.

4.4. BRIDGES

Many organizations have multiple LANs and wish to connect them. LANs can be connected by devices called **bridges**, which operate in the data link layer. This statement means that bridges do not examine the network layer header and can thus copy IP, IPX, and OSI packets equally well. In contrast, a pure IP, IPX, or OSI router can handle only its own native packets.

In the following sections we will look at bridge design, especially for connecting 802.3, 802.4, and 802.5 LANs. For a comprehensive treatment of bridges and related topics, see (Perlman, 1992). Before getting into the technology of bridges, it is worthwhile taking a look at some common situations in which bridges are used. We will mention six reasons why a single organization may end up with multiple LANs. First, many university and corporate departments have their own LANs, primarily to connect their own personal computers, workstations, and servers. Since the goals of the various departments differ, different departments choose different LANs, without regard to what other departments are doing. Sooner or later, there is a need for interaction, so bridges are needed. In this example, multiple LANs came into existence due to the autonomy of their owners.

Second, the organization may be geographically spread over several buildings separated by considerable distances. It may be cheaper to have separate LANs in each building and connect them with bridges and infrared links than to run a single coaxial cable over the entire site.

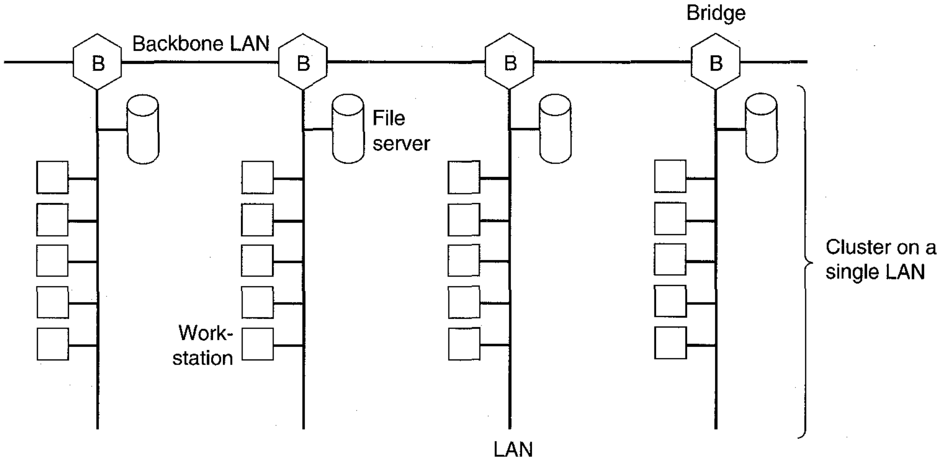


Fig. 4-34. Multiple LANs connected by a backbone to handle a total load higher than the capacity of a single LAN.

Third, it may be necessary to split what is logically a single LAN into separate LANs to accommodate the load. At many universities, for example, thousands of workstations are available for student and faculty computing. Files are normally kept on file server machines, and are downloaded to users' machines upon request. The enormous scale of this system precludes putting all the workstations on a single LAN—the total bandwidth needed is far too high. Instead multiple LANs connected by bridges are used, as shown in Fig. 4-34. Each LAN contains a cluster of workstations with its own file server, so that most traffic is restricted to a single LAN and does not add load to the backbone.

Fourth, in some situations, a single LAN would be adequate in terms of the load, but the physical distance between the most distant machines is too great (e.g., more than 2.5 km for 802.3). Even if laying the cable is easy to do, the network would not work due to the excessively long round-trip delay. The only solution is to partition the LAN and install bridges between the segments. Using bridges, the total physical distance covered can be increased.

Fifth, there is the matter of reliability. On a single LAN, a defective node that keeps outputting a continuous stream of garbage will cripple the LAN. Bridges can be inserted at critical places, like fire doors in a building, to prevent a single node which has gone berserk from bringing down the entire system. Unlike a repeater, which just copies whatever it sees, a bridge can be programmed to exercise some discretion about what it forwards and what it does not forward.

Sixth, and last, bridges can contribute to the organization's security. Most LAN interfaces have a **promiscuous mode**, in which *all* frames are given to the computer, not just those addressed to it. Spies and busybodies love this feature. By inserting bridges at various places and being careful not to forward sensitive traffic, it is possible to isolate parts of the network so that its traffic cannot escape and fall into the wrong hands.

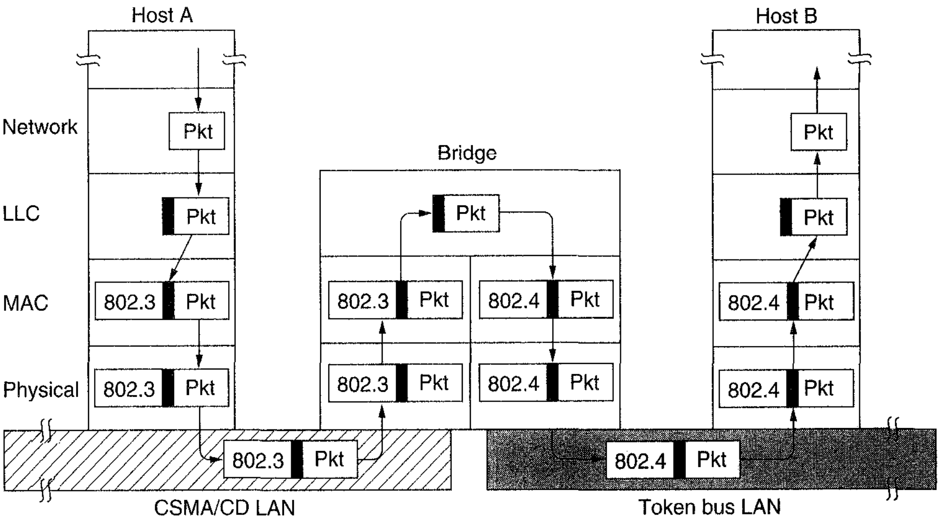


Fig. 4-35. Operation of a LAN bridge from 802.3 to 802.4.

Having seen why bridges are needed, let us now turn to the question of how they work. Figure 4-35 illustrates the operation of a simple two-port bridge. Host *A* has a packet to send. The packet descends into the LLC sublayer and acquires an LLC header. Then it passes into the MAC sublayer and an 802.3 header is prepended to it (also a trailer, not shown in the figure). This unit goes out onto the cable and eventually is passed up to the MAC sublayer in the bridge, where the 802.3 header is stripped off. The bare packet (with LLC header) is then handed off to the LLC sublayer in the bridge. In this example, the packet is destined for an 802.4 subnet connected to the bridge, so it works its way down the 802.4 side of the bridge and off it goes. Note that a bridge connecting *k* different LANs will have *k* different MAC sublayers and *k* different physical layers, one for each type.

4.4.1. Bridges from 802.x to 802.y

You might naively think that a bridge from one 802 LAN to another one would be completely trivial. Such is not the case. In the remainder of this section we will point out some of the difficulties that will be encountered when trying to build a bridge between the various 802 LANs.

Each of the nine combinations of 802.x to 802.y has its own unique set of problems. However, before dealing with these one at a time, let us look at some general problems common to all the bridges. To start with, each of the LANs uses a different frame format (see Fig. 4-36). There is no valid technical reason for this incompatibility. It is just that none of the corporations supporting the three standards (Xerox, GM, and IBM) wanted to change *theirs*. As a result, any copying between different LANs requires reformatting, which takes CPU time, requires a new checksum calculation, and introduces the possibility of undetected errors due to bad bits in the bridge's memory. None of this would have been necessary if the three committees had been able to agree on a single format.

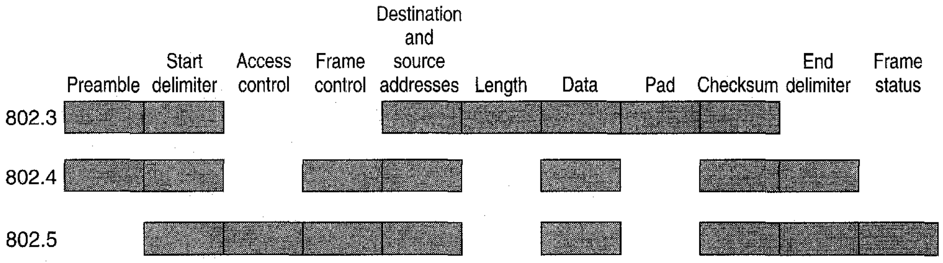


Fig. 4-36. The IEEE 802 frame formats.

A second problem is that interconnected LANs do not necessarily run at the same data rate. When forwarding a long run of back-to-back frames from a fast LAN to a slower one, the bridge will not be able to get rid of the frames as fast as they come in. It will have to buffer them, hoping not to run out of memory. The problem also exists from 802.4 to 802.3 at 10 Mbps to some extent because some of 802.3's bandwidth is lost to collisions. It does not really have 10 Mbps, whereas 802.4 really does (well, almost). Bridges that connect three or more LANs have a similar problem when several LANs are trying to feed the same output LAN at the same time.

A subtle, but important problem related to the bridge-as-bottleneck problem is the value of timers in the higher layers. Suppose that the network layer on an 802.4 LAN is trying to send a very long message as a sequence of frames. After sending the last one it starts a timer to wait for an acknowledgement. If the message has to transit a bridge to a slower 802.5 LAN, there is a danger that the timer will go off before the last frame has been forwarded onto the slower LAN. The network layer will assume the problem is due to a lost frame and just retransmit the entire sequence again. After *n* failed attempts it may give up and tell the transport layer that the destination is dead.

A third, and potentially most serious problem of all, is that all three 802 LANs have a different maximum frame length. For 802.3 it depends on the parameters of the configuration, but for the standard 10-Mbps system the payload is a maximum of 1500 bytes. For 802.4 it is fixed at 8191 bytes. For 802.5 there is no upper limit, except that a station may not transmit longer than the token-holding time. With the default value of 10 msec, the maximum frame length is 5000 bytes.

An obvious problem arises when a long frame must be forwarded onto a LAN that cannot accept it. Splitting the frame into pieces is out of the question in this layer. All the protocols assume that frames either arrive or they do not. There is no provision for reassembling frames out of smaller units. This is not to say that such protocols could not be devised. They could be and have been. It is just that 802 does not provide this feature. Basically, there is no solution. Frames that are too large to be forwarded must be discarded. So much for transparency.

Now let us briefly consider each of the nine cases of 802.x to 802.y bridges to see what other problems are lurking in the shadows. From 802.3 to 802.3 is easy. The only thing that can go wrong is that the destination LAN is so heavily loaded that frames keep pouring into the bridge, but the bridge cannot get rid of them. If this situation persists long enough, the bridge might run out of buffer space and begin dropping frames. Since this problem is always potentially present when forwarding onto an 802.3 LAN, we will not mention it further. With the other two LANs, each station, including the bridge is guaranteed to acquire the token periodically and cannot be shut out for long intervals.

From 802.4 to 802.3 two problems exist. First, 802.4 frames carry priority bits that 802.3 frames do not have. As a result, if two 802.4 LANs communicate via an 802.3 LAN, the priority will be lost by the intermediate LAN.

The second problem is caused by a specific feature in 802.4: temporary token handoff. It is possible for an 802.4 frame to have a header bit set to 1 to temporarily pass the token to the destination, to let it send an acknowledgement frame. However, if such a frame is forwarded by a bridge, what should the bridge do? If it sends an acknowledgement frame itself, it is lying because the frame really has not been delivered yet. In fact, the destination may be dead.

On the other hand, if it does not generate the acknowledgement, the sender will almost assuredly conclude that the destination is dead and report back failure to its superiors. There does not seem to be any way to solve this problem.

From 802.5 to 802.3 we have a similar problem. The 802.5 frame format has *A* and *C* bits in the frame status byte. These bits are set by the destination to tell the sender whether the station addressed saw the frame, and whether it copied it. Here again, the bridge can lie and say the frame has been copied, but if it later turns out that the destination is down, serious problems may arise. In essence, the insertion of a bridge into the network has changed the semantics of the bits. It is hard to imagine a proper solution to this problem.

From 802.3 to 802.4 we have the problem of what to put in the priority bits. A good case can be made for having the bridge retransmit all frames at the highest priority, because they have probably suffered enough delay already.

From 802.4 to 802.4 the only problem is what to do with the temporary token handoff. At least here we have the possibility of the bridge managing to forward the frame fast enough to get the response before the timer runs out. Still it is a gamble. By forwarding the frame at the highest priority, the bridge is telling a little white lie, but it thereby increases the probability of getting the response in time.

From 802.5 to 802.4 we have the same problem with the *A* and *C* bits as before. Also, the definition of the priority bits is different for the two LANs, but beggars can't be choosers. At least the two LANs have the same number of priority bits. All the bridge can do is copy the priority bits across and hope for the best.

From 802.3 to 802.5 the bridge must generate priority bits, but there are no other special problems. From 802.4 to 802.5 there is a potential problem with frames that are too long and the token handoff problem is present again. Finally, from 802.5 to 802.5 the problem is what to do with the *A* and *C* bits again. Figure 4-37 summarizes the various problems we have been discussing.

When the IEEE 802 committee set out to come up with a LAN standard, it was unable to agree on a single standard, so it produced *three* incompatible standards, as we have just seen in some detail. For this failure, it has been roundly criticized. When it was later assigned the job of designing a standard for bridges to interconnect its three incompatible LANs, it resolved to do better. It did. It came up with *two* incompatible bridge designs. So far nobody has asked it to design a gateway standard to connect its two incompatible bridges, but at least the trend is in the right direction.

This section has dealt with the problems encountered in connecting two IEEE 802 LANs via a single bridge. The next two sections deal with the problems of connecting large internetworks containing many LANs and many bridges and the two IEEE approaches to designing these bridges.

		Destination LAN		
		802.3 (CSMA/CD)	802.4 (Token bus)	802.4 (Token ring)
Source LAN	802.3		1, 4	1, 2, 4, 8
	802.4	1, 5, 8, 9, 10	9	1, 2, 3, 8, 9, 10
	802.5	1, 2, 5, 6, 7, 10	1, 2, 3, 6, 7	6, 7

Actions:

1. Reformat the frame and compute new checksum
2. Reverse the bit order.
3. Copy the priority, meaningful or not.
4. Generate a ficticious priority.
5. Discard priority.
6. Drain the ring (somehow).
7. Set A and C bits (by lying).
8. Worry about congestion (fast LAN to slow LAN).
9. Worry about token handoff ACK being delayed or impossible.
10. Panic if frame is too long for destination LAN.

Parameters assumed:

802.3:	1500-byte frames,	10 Mbps (minus collisions)
802.4:	8191-byte frames	10 Mbps
802.5:	5000-byte frames	4 Mbps

Fig. 4-37. Problems encountered in building bridges from 802.x to 802.y.

4.4.2. Transparent Bridges

The first 802 bridge is a **transparent bridge** or **spanning tree bridge** (Perlman, 1992). The overriding concern of the people who supported this design was complete transparency. In their view, a site with multiple LANs should be able to go out and buy bridges designed to the IEEE standard, plug the connectors into the bridges, and everything should work perfectly, instantly. There should be no hardware changes required, no software changes required, no setting of address switches, no downloading of routing tables or parameters, nothing. Just plug in the cables and walk away. Furthermore, the operation of the existing LANs should not be affected by the bridges at all. Surprisingly enough, they actually succeeded.

A transparent bridge operates in promiscuous mode, accepting every frame transmitted on all the LANs to which it is attached. As an example, consider the configuration of Fig. 4-38. Bridge B1 is connected to LANs 1 and 2, and bridge B2 is connected to LANs 2, 3, and 4. A frame arriving at bridge B1 on LAN 1 destined for *A* can be discarded immediately, because it is already on the right LAN, but a frame arriving on LAN 1 for *C* or *F* must be forwarded.

When a frame arrives, a bridge must decide whether to discard or forward it,

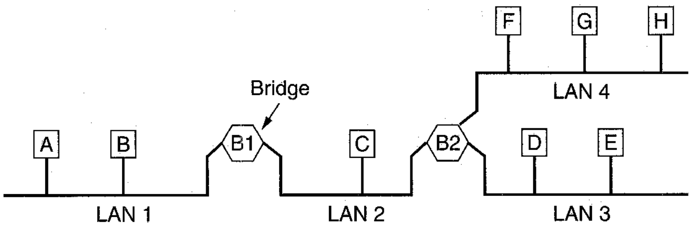


Fig. 4-38. A configuration with four LANs and two bridges.

and if the latter, on which LAN to put the frame. This decision is made by looking up the destination address in a big (hash) table inside the bridge. The table can list each possible destination and tell which output line (LAN) it belongs on. For example, B2's table would list *A* as belonging to LAN 2, since all B2 has to know is which LAN to put frames for *A* on. That, in fact, more forwarding happens later is not of interest to it.

When the bridges are first plugged in, all the hash tables are empty. None of the bridges know where any of the destinations are, so they use the flooding algorithm: every incoming frame for an unknown destination is output on all the LANs to which the bridge is connected except the one it arrived on. As time goes on, the bridges learn where destinations are, as described below. Once a destination is known, frames destined for it are put on only the proper LAN and are not flooded.

The algorithm used by the transparent bridges is **backward learning**. As mentioned above, the bridges operate in promiscuous mode, so they see every frame sent on any of their LANs. By looking at the source address, they can tell which machine is accessible on which LAN. For example, if bridge B1 in Fig. 4-38 sees a frame on LAN 2 coming from *C*, it knows that *C* must be reachable via LAN 2, so it makes an entry in its hash table noting that frames going to *C* should use LAN 2. Any subsequent frame addressed to *C* coming in on LAN 1 will be forwarded, but a frame for *C* coming in on LAN 2 will be discarded.

The topology can change as machines and bridges are powered up and down and moved around. To handle dynamic topologies, whenever a hash table entry is made, the arrival time of the frame is noted in the entry. Whenever a frame whose destination is already in the table arrives, its entry is updated with the current time. Thus the time associated with every entry tells the last time a frame from that machine was seen.

Periodically, a process in the bridge scans the hash table and purges all entries more than a few minutes old. In this way, if a computer is unplugged from its LAN, moved around the building, and replugged in somewhere else, within a few minutes it will be back in normal operation, without any manual intervention. This algorithm also means that if a machine is quiet for a few minutes, any traffic sent to it will have to be flooded, until it next sends a frame itself.

The routing procedure for an incoming frame depends on the LAN it arrives on (the source LAN) and the LAN its destination is on (the destination LAN), as follows:

1. If destination and source LANs are the same, discard the frame.
2. If the destination and source LANs are different, forward the frame.
3. If the destination LAN is unknown, use flooding.

As each frame arrives, this algorithm must be applied. Special purpose VLSI chips exist to do the lookup and update the table entry, all in a few microseconds.

To increase reliability, some sites use two or more bridges in parallel between pairs of LANs, as shown in Fig. 4-39. This arrangement, however, also introduces some additional problems because it creates loops in the topology.

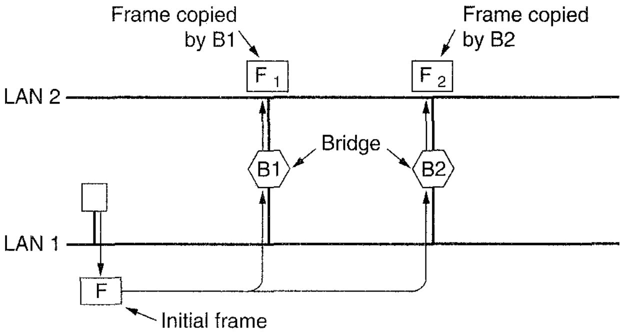


Fig. 4-39. Two parallel transparent bridges.

A simple example of these problems can be seen by observing how a frame, F, with unknown destination is handled in Fig. 4-39. Each bridge, following the normal rules for handling unknown destinations, uses flooding, which in this example, just means copying it to LAN 2. Shortly thereafter, bridge 1 sees F_2, a frame with an unknown destination, which it copies to LAN 1, generating F_3 (not shown). Similarly, bridge 2 copies F_1 to LAN 1 generating F_4 (also not shown). Bridge 1 now forwards F_4 and bridge 2 copies F_3. This cycle goes on forever.

Spanning Tree Bridges

The solution to this difficulty is for the bridges to communicate with each other and overlay the actual topology with a spanning tree that reaches every LAN. In effect, some potential connections between LANs are ignored in the interest of constructing a fictitious loop-free topology. For example, in Fig. 4-40(a) we see nine LANs interconnected by ten bridges. This configuration can be abstracted into a graph with the LANs as the nodes. An arc connects any two LANs that are connected by a bridge. The graph can be reduced to a spanning tree by dropping the arcs shown as dotted lines in Fig. 4-40(b). Using this spanning tree, there is exactly one path from every LAN to every other LAN. Once the bridges have agreed on the spanning tree, all forwarding between LANs follows the spanning tree. Since there is a unique path from each source to each destination, loops are impossible.

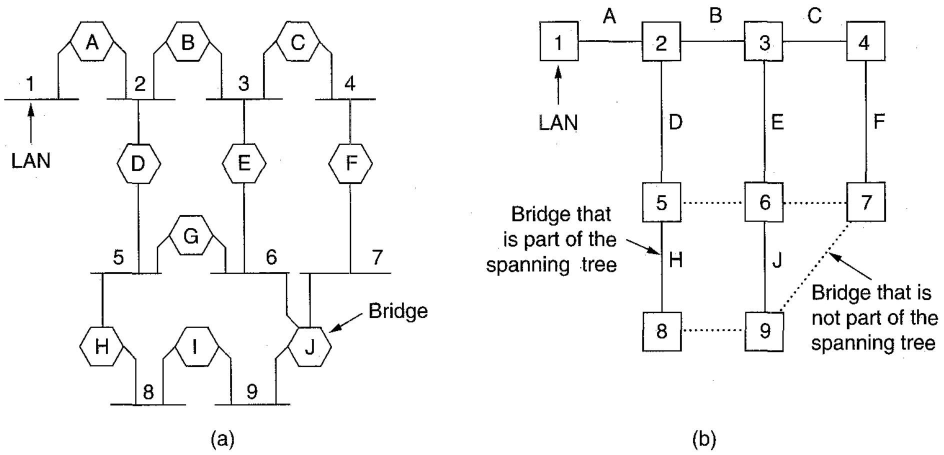

A
B
C
1
2
3
4
LAN
D
E
F
G
5
6
7
Bridge
H
I
J
8
9
(a)
Bridge that is part of the spanning tree
Bridge that is not part of the spanning tree
(b)

Fig. 4-40. (a) Interconnected LANs. (b) A spanning tree covering the LANs. The dotted lines are not part of the spanning tree.

To build the spanning tree, first the bridges have to choose one bridge to be the root of the tree. They make this choice by having each one broadcast its serial number, installed by the manufacturer, and guaranteed to be unique worldwide. The bridge with the lowest serial number becomes the root. Next, a tree of shortest paths from the root to every bridge and LAN is constructed. This tree is the spanning tree. If a bridge or LAN fails, a new one is computed.

The result of this algorithm is that a unique path is established from every LAN to the root, and thus to every other LAN. Although the tree spans all the LANs, not all the bridges are necessarily present in the tree (to prevent loops). Even after the spanning tree has been established, the algorithm continues to run in order to automatically detect topology changes and update the tree. The distributed algorithm used for constructing the spanning tree was invented by Perlman and is described in detail in (Perlman, 1992).

Bridges can also be used to connect LANs that are widely separated. In this model, each site consists of a collection of LANs and bridges, one of which has a connection to a WAN. Frames for remote LANs travel over the WAN. The basic spanning tree algorithm can be used, preferably with certain optimizations to select a tree that minimizes the amount of WAN traffic.

4.4.3. Source Routing Bridges

Transparent bridges have the advantage of being easy to install. You just plug them in and walk away. On the other hand, they do not make optimal use of the bandwidth, since they only use a subset of the topology (the spanning tree). The relative importance of these two (and other) factors led to a split within the 802 committees (Pitt, 1988). The CSMA/CD and token bus people chose the transparent bridge. The ring people (with encouragement from IBM) preferred a scheme called **source routing**, which we will now describe. For additional details, see (Dixon, 1987).

Reduced to its barest essentials, source routing assumes that the sender of each frame knows whether or not the destination is on its own LAN. When sending a frame to a different LAN, the source machine sets the high-order bit of the source address to 1, to mark it. Furthermore, it includes in the frame header the exact path that the frame will follow.

This path is constructed as follows. Each LAN has a unique 12-bit number, and each bridge has a 4-bit number that uniquely identifies it in the context of its LANs. Thus, two bridges far apart may both have number 3, but two bridges between the same two LANs must have different bridge numbers. A route is then a sequence of bridge, LAN, bridge, LAN, ... numbers. Referring to Fig. 4-38, the route from *A* to *D* would be (L1, B1, L2, B2, L3).

A source routing bridge is only interested in those frames with the high-order bit of the destination set to 1. For each such frame that it sees, it scans the route looking for the number of the LAN on which the frame arrived. If this LAN number is followed by its own bridge number, the bridge forwards the frame onto the LAN whose number follows its bridge number in the route. If the incoming LAN number is followed by the number of some other bridge, it does not forward the frame.

This algorithm lends itself to three possible implementations:

1. Software: the bridge runs in promiscuous mode, copying all frames to its memory to see if they have the high-order destination bit set to 1. If so, the frame is inspected further; otherwise it is not.

2. Hybrid: the bridge's LAN interface inspects the high-order destination bit and only accepts frames with the bit set. This interface is easy to build into hardware and greatly reduces the number of frames the bridge must inspect.

3. Hardware: the bridge's LAN interface not only checks the high-order destination bit, but it also scans the route to see if this bridge must do forwarding. Only frames that must actually be forwarded are given to the bridge. This implementation requires the most complex hardware but wastes no bridge CPU cycles because all irrelevant frames are screened out.

These three implementations vary in their cost and performance. The first one has no additional hardware cost for the interface but may require a very fast CPU to handle all the frames. The last one requires a special VLSI chip but offloads much of the processing from the bridge to the chip, so that a slower CPU can be used, or alternatively, the bridge can handle more LANs.

Implicit in the design of source routing is that every machine in the internetwork knows, or can find, the best path to every other machine. How these routes are discovered is an important part of the source routing algorithm. The basic idea is that if a destination is unknown, the source issues a broadcast frame asking where it is. This **discovery frame** is forwarded by every bridge so that it reaches every LAN on the internetwork. When the reply comes back, the bridges record their identity in it, so that the original sender can see the exact route taken and ultimately choose the best route.

While this algorithm clearly finds the best route (it finds *all* routes), it suffers from a frame explosion. Consider the configuration of Fig. 4-41, with N LANs linearly connected by triple bridges. Each discovery frame sent by station 1 is copied by each of the three bridges on LAN 1, yielding three discovery frames on LAN 2. Each of these is copied by each of the bridges on LAN 2, resulting in nine frames on LAN 3. By the time we reach LAN N, 3^{N-1} frames are circulating. If a dozen sets of bridges are traversed, more than half a million discovery frames will have to be injected into the last LAN, causing severe congestion.

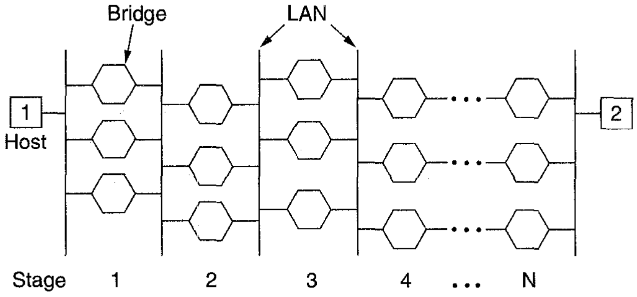


Fig. 4-41. A series of LANs connected by triple bridges.

A somewhat analogous process happens with the transparent bridge, only it is not nearly so severe. When an unknown frame arrives, it is flooded, but only along the spanning tree, so the total volume of frames sent is linear with the size of the network, not exponential.

Once a host has discovered a route to a certain destination, it stores the route in a cache, so that the discovery process will not have to be run next time. While this approach greatly limits the impact of the frame explosion, it does put some administrative burden on all the hosts, and the whole algorithm is definitely not transparent, which was one of the original goals, as we mentioned above.

4.4.4. Comparison of 802 Bridges

The transparent and source routing bridges each have advantages and disadvantages. In this section we will discuss some of the major ones. They are summarized in Fig. 4-42 and covered in more detail in (Soha and Perlman, 1988; and Zhang, 1988). Be warned, however, that every one of the points is highly contested.

Issue	Transparent bridge	Source routing bridge
Orientation	Connectionless	Connection-oriented
Transparency	Fully transparent	Not transparent
Configuration	Automatic	Manual
Routing	Suboptimal	Optimal
Locating	Backward learning	Discovery frames
Failures	Handled by the bridges	Handled by the hosts
Complexity	In the bridges	In the hosts

Fig. 4-42. Comparison of transparent and source routing bridges.

At the heart of the difference between the two bridge types is the distinction between connectionless and connection-oriented networking. The transparent bridges have no concept of a virtual circuit at all and route each frame independently from all the others. The source routing bridges, in contrast, determine a route using discovery frames and then use that route thereafter.

The transparent bridges are completely invisible to the hosts and are fully compatible with all existing 802 products. The source routing bridges are neither transparent nor compatible. To use source routing, hosts must be fully aware of the bridging scheme and must actively participate in it. Splitting an existing LAN into two LANs connected by a source routing bridge requires making changes to the host software.

When using transparent bridges, no network management is needed. The bridges configure themselves to the topology automatically. With source routing bridges, the network manager must manually install the LAN and bridge numbers. Mistakes, such as duplicating a LAN or bridge number, can be very difficult to detect, as they may cause some frames to loop, but not others on different routes. Furthermore, when connecting two previously disjoint internetworks, with transparent bridges there is nothing to do except connect them, whereas with source routing, it may be necessary to manually change many LAN numbers to make them unique in the combined internetwork.

One of the few advantages of source routing is that, in theory, it can use optimal routing, whereas transparent bridging is restricted to the spanning tree.

Furthermore, source routing can also make good use of parallel bridges between two LANs to split the load. Whether actual bridges will be clever enough to make use of these theoretical advantages is questionable.

Locating destinations is done using backward learning in the transparent bridge and using discovery frames in source routing bridges. The disadvantage of backward learning is that the bridges have to wait until a frame from a particular machine happens to come along in order to learn where that machine is. The disadvantage of discovery frames is the exponential explosion in moderate to large internetworks with parallel bridges.

Failure handling is quite different in the two schemes. Transparent bridges learn about bridge and LAN failures and other topology changes quickly and automatically, just from listening to each other's control frames. Hosts do not notice these changes at all.

With source routing, the situation is quite different. When a bridge fails, machines that are routing over it initially notice that their frames are no longer being acknowledged, so they time out and try over and over. Finally, they conclude that something is wrong, but they still do not know if the problem is with the destination itself, or with the current route. Only by sending another discovery frame can they see if the destination is available. Unfortunately, when a major bridge fails, a large number of hosts will have to experience timeouts and send new discovery frames before the problem is resolved, even if an alternative route is available. This greater vulnerability to failures is one of the major weaknesses of all connection-oriented systems.

Finally, we come to complexity and cost, a very controversial topic. If source routing bridges have a VLSI chip that reads in only those frames that must be forwarded, these bridges will experience a lighter frame processing load and deliver a better performance for a given investment in hardware. Without this chip they will do worse because the amount of processing per frame (searching the route in the frame header) is substantially more.

In addition, source routing puts extra complexity in the hosts: they must store routes, send discovery frames, and copy route information into each frame. All of these things require memory and CPU cycles. Since there are typically one to two orders of magnitude more hosts than bridges, it may be better to put the extra cost and complexity into a few bridges, rather than in all the hosts.

4.4.5. Remote Bridges

A common use of bridges is to connect two (or more) distant LANs. For example, a company might have plants in several cities, each with its own LAN. Ideally, all the LANs should be interconnected, so the complete system acts like one large LAN.

This goal can be achieved by putting a bridge on each LAN and connecting the bridges pairwise with point-to-point lines (e.g., lines leased from a telephone company). A simple system, with three LANs, is illustrated in Fig. 4-43. The usual routing algorithms apply here. The simplest way to see this is to regard the three point-to-point lines as hostless LANs. Then we have a normal system of six LANS interconnected by four bridges. Nothing in what we have studied so far says that a LAN must have hosts on it.

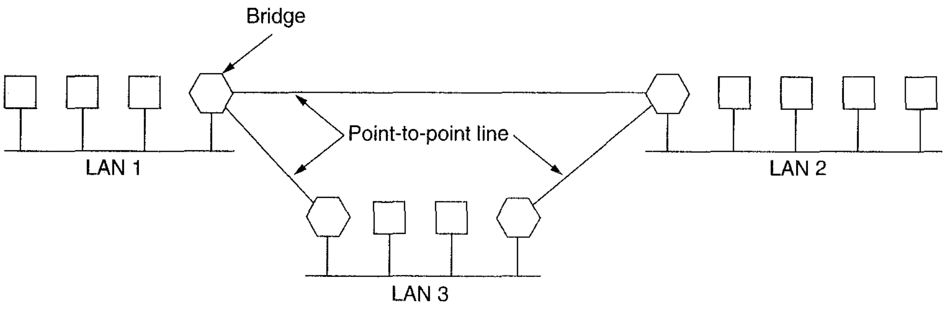


Fig. 4-43. Remote bridges can be used to interconnect distant LANs.

Various protocols can be used on the point-to-point lines. One possibility is to choose some standard point-to-point data link protocol, putting complete MAC frames in the payload field. This strategy works best if all the LANs are identical, and the only problem is getting frames to the right LAN. Another option is to strip off the MAC header and trailer at the source bridge and put what is left in the payload field of the point-to-point protocol. A new MAC header and trailer can then be generated at the destination bridge. A disadvantage of this approach is that the checksum that arrives at the destination host is not the one computed by the source host, so errors caused by bad bits in a bridge's memory may not be detected.

4.5. HIGH-SPEED LANS

The 802 LANs and MAN we have just studied are all based on one copper wire (two copper wires for 802.6). For low speeds and short distances, this will do just fine, but for high speeds and longer distances LANs must be based on fiber optics or highly parallel copper networks. Fiber has high bandwidth, is thin and lightweight, is not affected by electromagnetic interference from heavy machinery (important when cabling runs through elevator shafts), power surges, or lightning, and has excellent security because it is nearly impossible to wiretap without detection. Consequently, fast LANs often use fiber. In the following sections we will look at some local area networks that use fiber optics, as well as one extremely high-speed LAN that uses old fashioned copper wire (but lots of it).

4.5.1. FDDI

FDDI (Fiber Distributed Data Interface) is a high-performance fiber optic token ring LAN running at 100 Mbps over distances up to 200 km with up to 1000 stations connected (Black, 1994; Jain, 1994; Mirchandani and Khanna, 1993; Ross and Hamstra, 1993; Shah and Ramakrishnan, 1994; and Wolter, 1990). It can be used in the same way as any of the 802 LANs, but with its high bandwidth, another common use is as a backbone to connect copper LANs, as shown in Fig. 4-44. FDDI-II is the successor to FDDI, modified to handle synchronous circuit-switched PCM data for voice or ISDN traffic, in addition to ordinary data. We will refer to both of them as just FDDI. This section deals with both the physical layer and the MAC sublayer of FDDI.

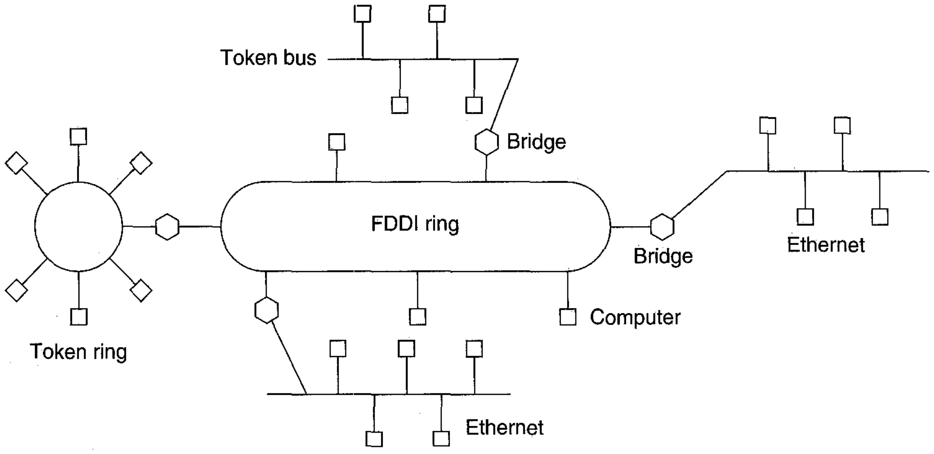


Fig. 4-44. An FDDI ring being used as a backbone to connect LANs and computers.

FDDI uses multimode fibers because the additional expense of single mode fibers is not needed for networks running at only 100 Mbps. It also uses LEDs rather than lasers, not only due to their lower cost, but also because FDDI may sometimes be used to connect directly to user workstations. There is a danger that curious users may occasionally unplug the fiber connector and look directly into it to watch the bits go by at 100 Mbps. With a laser the curious user might end up with a hole in his retina. LEDs are too weak to do any eye damage but are strong enough to transfer data accurately at 100 Mbps. The FDDI design specification calls for no more than 1 error in 2.5×10^{10} bits. Many implementations do much better.

The FDDI cabling consists of two fiber rings, one transmitting clockwise and the other transmitting counterclockwise, as illustrated in Fig. 4-45(a). If either one breaks, the other can be used as a backup. If both break at the same point, for example, due to a fire or other accident in the cable duct, the two rings can be joined into a single ring approximately twice as long, as shown in Fig. 4-45(b). Each station contains relays that can be used to join the two rings or bypass the station in the event of station problems. Wire centers can also be used, as in 802.5.

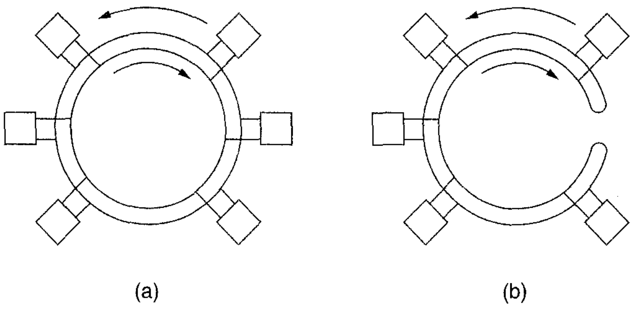


Fig. 4-45. (a) FDDI consists of two counterrotating rings. (b) In the event of failure of both rings at one point, the two rings can be joined together to form a single long ring.

FDDI defines two classes of stations, *A* and *B*. Class *A* stations connect to both rings. The cheaper class *B* stations only connect to one of the rings. Depending on how important fault tolerance is, an installation can choose class *A* or class *B* stations, or some of each.

The physical layer does not use Manchester encoding because 100-Mbps Manchester encoding requires 200 megabaud, which was deemed too expensive. Instead a scheme called **4 out of 5** encoding is used. Each group of 4 MAC symbols (0s, 1s, and certain nondata symbols such as start-of-frame) are encoded as a group of 5 bits on the medium. Sixteen of the 32 combinations are for data, 3 are for delimiters, 2 are for control, 3 are for hardware signaling, and 8 are unused (i.e., reserved for future versions of the protocol).

The advantage of this scheme is that it saves bandwidth, but the disadvantage is the loss of the self-clocking property of Manchester encoding. To compensate for this loss, a long preamble is used to synchronize the receiver to the sender's clock. Furthermore, all clocks are required to be stable to at least 0.005 percent. With this stability, frames up to 4500 bytes can be sent without danger of the receiver's clock drifting too far out of sync with the data stream.

The basic FDDI protocols are closely modeled on the 802.5 protocols. To transmit data, a station must first capture the token. Then it transmits a frame and removes it when it comes around again. One difference between FDDI and 802.5 is that in 802.5, a station may not generate a new token until its frame has gone all the way around and come back. In FDDI, with potentially 1000 stations and 200 km of fiber, the amount of time wasted waiting for the frame to circumnavigate the ring could be substantial. For this reason, it was decided to allow a station to put a new token back onto the ring as soon as it has finished transmitting its frames. In a large ring, several frames might be on the ring at the same time.

FDDI data frames are similar to 802.5 data frames. The FDDI format is shown in Fig. 4-46. The *Start delimiter* and *End delimiter* fields mark the frame boundaries. The *Frame control* field tells what kind of frame this is (data, control, etc.). The *Frame status* byte holds acknowledgement bits, similar to those of 802.5. The other fields are analogous to 802.5.

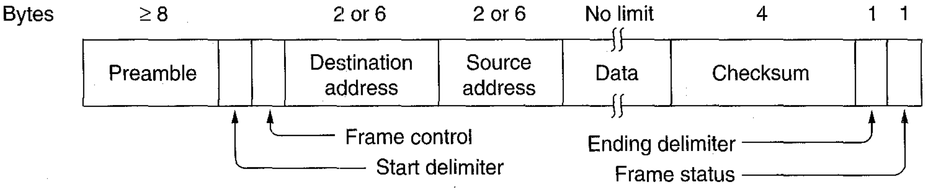


Fig. 4-46. FDDI frame format.

In addition to the regular (asynchronous) frames, FDDI also permits special synchronous frames for circuit-switched PCM or ISDN data. The synchronous frames are generated every 125 μsec by a master station to provide the 8000 samples/sec needed by PCM systems. Each of these frames has a header, 16 bytes of noncircuit-switched data, and up to 96 bytes of circuit-switched data (i.e., up to 96 PCM channels per frame).

The number 96 was chosen because it allows four T1 channels (4 × 24) at 1.544 Mbps or three CCITT E1 channels (3 × 32) at 2.048 Mbps to fit in a frame, thus making it suitable for use anywhere in the world. One synchronous frame every 125 μsec consumes 6.144 Mbps of bandwidth for the 96 circuit-switched channels. A maximum of 16 synchronous frames every 125 μsec allows up to 1536 PCM channels and eats up 98.3 Mbps.

Once a station has acquired one or more time slots in a synchronous frame, those slots are reserved for it until they are explicitly released. The total bandwidth not used by the synchronous frames is allocated on demand. A bit mask is present in each of these frames to indicate which slots are available for demand assignment. The nonsynchronous traffic is divided into priority classes, with the higher priorities getting first shot at the leftover bandwidth.

The FDDI MAC protocol uses three timers. The **token holding timer** determines how long a station may continue to transmit once it has acquired the token. This timer prevents a station from hogging the ring forever. The **token rotation timer** is restarted every time the token is seen. If this timer expires, it means that the token has not been sighted for too long an interval. Probably it has been lost, so the token recovery procedure is initiated. Finally, the **valid transmission timer** is used to time out and recover from certain transient ring errors.

FDDI also has a priority algorithm similar to 802.4. It determines which priority classes may transmit on a given token pass. If the token is ahead of schedule, all priorities may transmit, but if it is behind schedule, only the highest ones may send.

4.5.2. Fast Ethernet

FDDI was supposed to be the next generation LAN, but it never really caught on much beyond the backbone market (where it continues to do fine). The station management was too complicated, which led to complex chips and high prices. The substantial cost of FDDI chips made workstation manufacturers unwilling to make FDDI the standard network, so volume production never happened and FDDI never broke through to the mass market. The lesson that should have been learned here was KISS (Keep It Simple, Stupid).

In any event, the failure of FDDI to catch fire left a gap for a garden-variety LAN at speeds above 10 Mbps. Many installations needed more bandwidth and thus had numerous 10-Mbps LANs connected by a maze of repeaters, bridges, routers, and gateways, although to the network managers it sometimes felt that they were being held together by bubble gum and chicken wire.

It was in this environment that IEEE reconvened the 802.3 committee in 1992 with instructions to come up with a faster LAN. One proposal was to keep 802.3 exactly as it was, but just make it go faster. Another proposal was to redo it totally, to give it lots of new features, such as real-time traffic and digitized voice, but just keep the old name (for marketing reasons). After some wrangling, the committee decided to keep 802.3 the way it was, but just make it go faster. The people behind the losing proposal did what any computer-industry people would have done under these circumstances—they formed their own committee and standardized their LAN anyway (eventually as 802.12).

The three primary reasons that the 802.3 committee decided to go with a souped-up 802.3 LAN were:

1. The need to be backward compatible with thousands of existing LANs.
2. The fear that a new protocol might have unforeseen problems.
3. The desire to get the job done before the technology changed.

The work was done quickly (by standards committees' norms), and the result, **802.3u**, was officially approved by IEEE in June 1995. Technically, 802.3u is not a new standard, but an addendum to the existing 802.3 standard (to emphasize its backward compatibility). Since everyone calls it **fast Ethernet**, rather than 802.3u, we will do that, too.

The basic idea behind fast Ethernet was simple: keep all the old packet formats, interfaces, and procedural rules, but just reduce the bit time from 100 nsec to 10 nsec. Technically, it would have been possible to copy 10Base-5 or 10Base-2 and still detect collisions on time by just reducing the maximum cable length by a factor of ten. However, the advantages of 10Base-T wiring were so overwhelming, that fast Ethernet is based entirely on this design. Thus all fast Ethernet systems use hubs; multidrop cables with vampire taps or BNC connectors are not permitted.

Nevertheless, some choices still had to be made, the most important of which was which wire types to support. One contender was category 3 twisted pair. The argument for it was that practically every office in the Western world has at least four category 3 (or better) twisted pairs running from it to a telephone wiring closet within 100 meters. Sometimes two such cables exist. Thus using category 3 twisted pair would make it possible to wire up desktop computers using fast Ethernet without having to rewire the building, an enormous advantage for many organizations.

The main disadvantage of category 3 twisted pair is its inability to carry 200 megabaud signals (100 Mbps with Manchester encoding) 100 meters, the maximum computer-to-hub distance specified for 10Base-T (see Fig. 4-17). In contrast, category 5 twisted pair wiring can handle 100 meters easily, and fiber can go much further. The compromise chosen was to allow all three possibilities, as shown in Fig. 4-47, but to pep up the category 3 solution to give it the additional carrying capacity needed.

Name	Cable	Max. segment	Advantages
100Base-T4	Twisted pair	100 m	Uses category 3 UTP
100Base-TX	Twisted pair	100 m	Full duplex at 100 Mbps
100Base-FX	Fiber optics	2000 m	Full duplex at 100 Mbps; long runs

Fig. 4-47. Fast Ethernet cabling.

The category 3 UTP scheme, called **100Base-T4**, uses a signaling speed of 25 MHz, only 25 percent faster than standard 802.3's 20 MHz (remember that Manchester encoding, as shown in Fig. 4-20, requires two clock periods for each of the 10 million bits each second). To achieve the necessary bandwidth, 100Base-T4 requires four twisted pairs. Since standard telephone wiring for decades has had four twisted pairs per cable, most offices are able to handle this. Of course, it means giving up your office telephone, but that is surely a small price to pay for faster email.

Of the four twisted pairs, one is always to the hub, one is always from the hub, and other two are switchable to the current transmission direction. To get the necessary bandwidth, Manchester encoding is not used, but with modern clocks and such short distances, it is no longer needed. In addition, ternary signals are sent, so that during a single clock period the wire can contain a 0, a 1, or a 2. With three twisted pairs going in the forward direction and ternary signaling, any one of 27 possible symbols can be transmitted, making it possible to send 4 bits with some redundancy. Transmitting 4 bits in each of the 25 million clock cycles per second gives the necessary 100 Mbps. In addition, there is always a 33.3 Mbps reverse channel using the remaining twisted pair. This scheme, known as **8B6T**, (8 bits map to 6 trits) is not likely to win any prizes for elegance, but it works with the existing wiring plant.

For category 5 wiring, the design, **100Base-TX**, is simpler because the wires can handle clock rates up to 125 MHz and beyond. Only two twisted pairs per station are used, one to the hub and one from it. Rather than just use straight binary coding, a scheme called **4B5B** is used at 125 MHz. Every group of five clock periods is used to send 4 bits in order to give some redundancy, provide enough transitions to allow easy clock synchronization, create unique patterns for frame delimiting, and be compatible with FDDI in the physical layer. Consequently, 100Base-TX is a full-duplex system; stations can transmit at 100 Mbps and receive at 100 Mbps at the same time. In addition, you can have two telephones in your office for real communication in case the computer is fully occupied with surfing the Web.

The last option, **100Base-FX**, uses two strands of multimode fiber, one for each direction, so it, too, is full duplex with 100 Mbps in each direction. In addition, the distance between a station and the hub can be up to 2 km.

Two kinds of hubs are possible with 100Base-T4 and 100Base-TX, collectively known as **100Base-T**. In a shared hub, all the incoming lines (or at least all the lines arriving at one plug-in card) are logically connected, forming a single collision domain. All the standard rules, including the binary backoff algorithm, apply, so the system works just like old-fashioned 802.3. In particular, only one station at a time can be transmitting.

In a switched hub, each incoming frame is buffered on a plug-in line card. Although this feature makes the hub and cards more expensive, it also means that all stations can transmit (and receive) at the same time, greatly improving the total bandwidth of the system, often by an order of magnitude or more. Buffered frames are passed over a high-speed backplane from the source card to the destination card. The backplane has not been standardized, nor does it need to be, since it is entirely hidden deep inside the switch. If past experience is any guide, switch vendors will compete vigorously to produce ever faster backplanes in order to improve system throughput. Because 100Base-FX cables are too long for the normal Ethernet collision algorithm, they must be connected to buffered, switched hubs, so each one is a collision domain unto itself.

As a final note, virtually all switches can handle a mix of 10-Mbps and 100-Mbps stations, to make upgrading easier. As a site acquires more and more 100-Mbps workstations, all it has to do is buy the necessary number of new line cards and insert them into the switch.

More information about Fast Ethernet can be found in (Johnson, 1996). For a comparison of high-speed local area networks, in particular, FDDI, fast Ethernet, ATM, and VG-AnyLAN, see (Cronin et al., 1994).

4.5.3. HIPPI—High-Performance Parallel Interface

During the Cold War, Los Alamos National Laboratory, the U.S. government's nuclear weapons design center, routinely bought one of every supercomputer offered for sale. Los Alamos also collected fancy peripherals, such as massive storage devices and special graphics workstations for scientific visualization. At that time, each manufacturer had a different interface for connecting peripherals to its supercomputer, so it was not possible to share peripherals among machines or to connect two supercomputers together.

In 1987, researchers at Los Alamos began work on a standard supercomputer interface, with the intention of getting it standardized and then talking all the vendors into using it. (Given the size of Los Alamos' computing budget, when it talked, vendors listened.) The goal for the interface was an interface that everyone could implement quickly and efficiently. The guiding principle was KISS: Keep It Simple, Stupid. It was to have no options, not require any new chips to be designed, and have the performance of a fire hose.

The initial specification called for a data rate of 800 Mbps, because watching movies of bombs going off required frames of 1024 × 1024 pixels with 24 bits per pixel and 30 frames/sec, for an aggregate data rate of 750 Mbps. Later, one option crept in: a second data rate of 1600 Mbps. When this proposal, called **HIPPI** (**HIgh Performance Parallel Interface**) was later offered to ANSI for standardization, the proposers were regarded as the lunatic fringe because LANs in the 1980s meant 10-Mbps Ethernets.

HIPPI was originally designed to be a data channel rather than a LAN. Data channels operate point-to-point, from one master (a computer) to one slave (a peripheral), with dedicated wires and no switching. No contention is present and the environment is entirely predictable. Later, the need to be able to switch a peripheral from one supercomputer to another became apparent, and a crossbar switch was added to the HIPPI design, as illustrated in Fig. 4-48.

In order to achieve such enormous performance using only off-the-shelf chips, the basic interface was made 50 bits wide, 32 bits of data and 18 bits of control, so the HIPPI cable contains 50 twisted pairs. Every 40 nsec, a word is transferred in parallel across the interface. To achieve 1600 Mbps, two cables are used and two words are transferred per cycle. All transfers are simplex. To get two-way communication, two (or four) cables are needed. At these speeds, the maximum cable length is 25 meters.

After it got over some initial shock, the ANSI X3T9.3 committee produced a HIPPI standard based on the Los Alamos input. The standard covers the physical and data link layers. Everything above that is up to the users. The basic protocol is that to communicate, a host first asks the crossbar switch to set up a connection. Then it (usually) sends a single message and releases the connection.

Messages are structured with a control word, a header of up to 1016 bytes, and a data part of up to $2^{32} - 2$ bytes. For flow control reasons, messages are

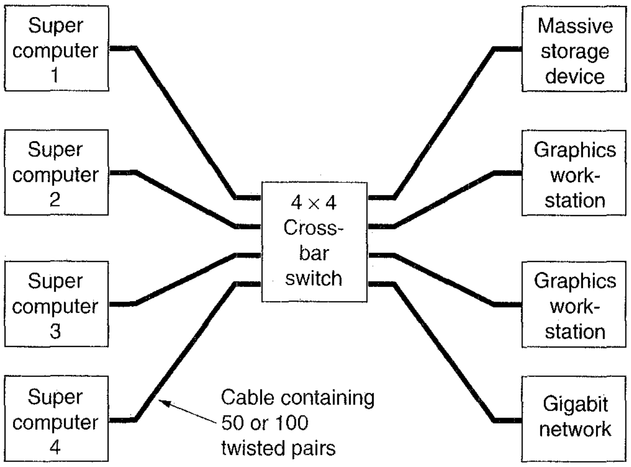


Fig. 4-48. HIPPI using a crossbar switch.

broken up into frames of 256 words. When the receiver is able to accept a frame, it signals the sender, which then sends a frame. Receivers can also ask for multiple frames at once. Error control consists of a horizontal parity bit per word and a vertical parity word at the end of each frame. Traditional checksums were regarded as unnecessary and too slow.

HIPPI was quickly implemented by dozens of vendors and has been the supercomputer interconnect standard for years. For more information about it, see (Hughes and Franta, 1994; Tolmie, 1992; and Tolmie and Renwick, 1993).

4.5.4. Fibre Channel

At the time HIPPI was designed, fiber optics was too expensive and not considered reliable enough, so the fastest LAN ever built was constructed from low-grade telephone wire. As time went on, fiber became cheaper and more reliable, so it was natural that there would eventually be an attempt to redo HIPPI using a single fiber instead of 50 or 100 twisted pairs. Unfortunately, the discipline that Los Alamos had in beating down proposed new features every time one reared its ugly head was lost along the way. The successor to HIPPI, called **fibre channel**, is far more complicated and more expensive to implement. Whether it will enjoy HIPPI's commercial success remains to be seen.

Fibre channel handles both data channel and network connections. In particular, it can be used to carry data channels including HIPPI, SCSI, and the multiplexor channel used on IBM mainframes. It can also carry network packets, including IEEE 802, IP, and ATM. Like HIPPI, the basic structure of fibre channel is a crossbar switch that connects inputs to outputs. Connections can be established for a single packet or for a much longer interval.

Fibre channel supports three service classes. The first class is pure circuit switching, with guaranteed delivery in order. The data channel modes use this service class. The second class is packet switching with guaranteed delivery. The third class is packet switching without guaranteed delivery.

Fibre channel has an elaborate protocol structure, as shown in Fig. 4-49. Here we see five layers, which together cover the physical and data link layers. The bottom layer deals with the physical medium. So far, it supports data rates of 100, 200, 400, and 800 Mbps. The second layer handles the bit encoding. The system used is somewhat like FDDI, but instead of 5 bits being used to encode 16 valid symbols, 10 bits are used to encode 256 valid symbols, providing a small amount of redundancy. Together, these two layers are functionally equivalent to the OSI physical layer.

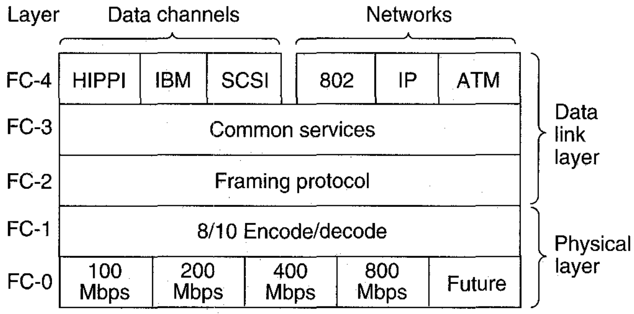


Fig. 4-49. The fibre channel protocol layers.

The middle layer defines the frame layout and header formats. Data are transmitted in frames whose payloads can be up to 2048 bytes. The next layer allows common services to be provided to the top layer in the future, as required. Finally, the top layer provides the interfaces to the various kinds of computers and peripherals supported.

As an aside, although fibre channel was designed in the United States, the spelling of the name was chosen by the editor of the standard, who was British. Additional information about fibre channel can be found in (Tolmie, 1992). A comparison of it with HIPPI and ATM is in (Tolmie, 1995).

4.6. SATELLITE NETWORKS

Although most multiple access channels are found in LANs, one kind of WAN also uses multiple access channels: communication satellite based WANs. In the following sections we will briefly study some of the problems that occur with satellite-based wide area networks. We will also look at some of the protocols that have been devised to deal with them.

Communication satellites generally have up to a dozen or so transponders. Each transponder has a beam that covers some portion of the earth below it, ranging from a wide beam 10,000 km across to a spot beam only 250 km across. Stations within the beam area can send frames to the satellite on the uplink frequency. The satellite then rebroadcasts them on the downlink frequency. Different frequencies are used for uplink and downlink to keep the transponder from going into oscillation. Satellites that do no on-board processing, but just echo whatever they hear (most of them), are often called **bent pipe** satellites.

Each antenna can aim itself at some area, transmit some frames, and then aim at a new area. Aiming is done electronically, but still takes some number of microseconds. The amount of time a beam is pointed to a given area is called the **dwell time**. For maximum efficiency, it should not be too short or too much time will be wasted moving the beam.

Just as with LANs, one of the key design issues is how to allocate the transponder channels. However, unlike LANs, carrier sensing is impossible due to the 270-msec propagation delay. When a station senses the state of a downlink channel, it hears what was going on 270 msec ago. Sensing the uplink channel is generally impossible. As a result, the CSMA/CD protocols (which assume that a transmitting station can detect collisions within the first few bit times, and then pull back if one is occurring) cannot be used with satellites. Hence the need for other protocols.

Five classes of protocols are used on the multiple access (uplink) channel: polling, ALOHA, FDM, TDM, and CDMA. Although we have studied each of these already, satellite operation sometimes adds new twists. The main problem is with the uplink channel, since the downlink channel has only a single sender (the satellite) and thus has no channel allocation problem.

4.6.1. Polling

The traditional way to allocate a single channel among competing users is for somebody to poll them. Having the satellite poll each station in turn to see if it has a frame is prohibitively expensive, given the 270-msec time required for each poll/response sequence.

However, if all the ground stations are also tied to a (typically low-bandwidth) packet-switching network, a minor variation of this idea is conceivable. The idea is to arrange all the stations in a logical ring, so each station knows its successor. Around this terrestrial ring circulates a token. The satellite never sees the token. A station is allowed to transmit on the uplink only when it has captured the token. If the number of stations is small and constant, the token transmission time is short, and the bursts sent on the uplink channel are much longer than the token rotation time, the scheme is moderately efficient.

4.6.2. ALOHA

Pure ALOHA is easy to implement: every station just sends whenever it wants to. The trouble is that the channel efficiency is only about 18 percent. Generally, such a low utilization factor is unacceptable for satellites that costs tens of millions of dollars each.

Using slotted ALOHA doubles the efficiency but introduces the problem of how to synchronize all the stations so they all know when each time slot begins. Fortunately, the satellite itself holds the answer, since it is inherently a broadcast medium. One ground station, the **reference station**, periodically transmits a special signal whose rebroadcast is used by all the ground stations as the time origin. If the time slots all have length ΔT, each station now knows that time slot k begins at a time $k \Delta T$ after the time origin. Since clocks run at slightly different rates, periodic resynchronization is necessary to keep everyone in phase. An additional complication is that the propagation time from the satellite is different for each ground station, but this effect can be corrected for.

To increase the utilization of the uplink channel above $1/e$, we could go from the single uplink channel of Fig. 4-50(a), to the dual uplink scheme of Fig. 4-50(b). A station with a frame to transmit chooses one of the two uplink channels at random and sends the frame in the next slot. Each uplink then operates an independent slotted ALOHA channel.

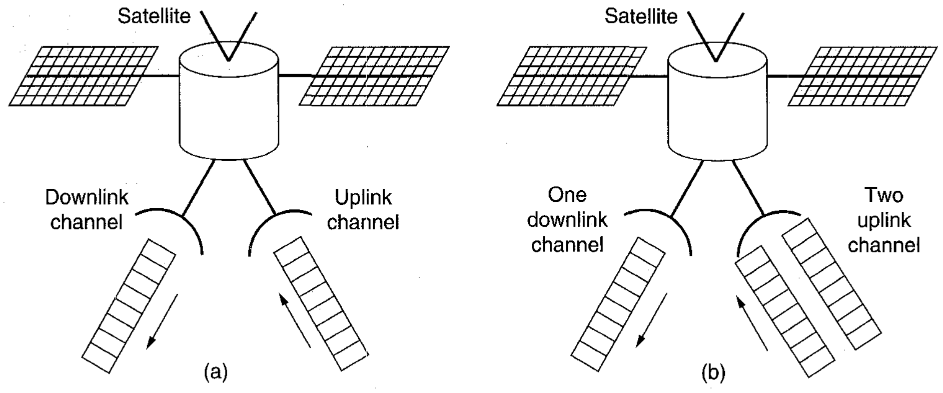


Fig. 4-50. (a) A standard ALOHA system. (b) Adding a second uplink channel.

If one of the uplink channels contains a single frame, it is just transmitted in the corresponding downlink slot later. If both channels are successful, the satellite can buffer one of the frames and transmit it during an idle slot later on. Working out the probabilities, it can be shown that given an infinite amount of buffer space, the downlink utilization can be gotten up to 0.736 at a cost of increasing the bandwidth requirements by one half.

4.6.3. FDM

Frequency division multiplexing is the oldest and probably still the most widely used channel allocation scheme. A typical 36-Mbps transponder might be divided statically into 500 or so 64,000-bps PCM channels, each one operating at its own unique frequency to avoid interfering with the others.

Although simple, FDM also has some drawbacks. First, guard bands are needed between the channels to keep the stations separated. This requirement exists because it is not possible to build transmitters that output all their energy in the main band and nothing in the side bands. The amount of bandwidth wasted in the guard bands can be a substantial fraction of the total.

Second, the stations must be carefully power controlled. If a station puts out too much power in the main band, it will also automatically put out too much power in the side bands, spilling over into adjacent channels and causing interference. Finally, FDM is entirely an analog technique and does not lend itself well to implementation in software.

If the number of stations is small and fixed, the frequency channels can be allocated statically in advance. However, if the number of stations, or the load on each one can fluctuate rapidly, some form of dynamic allocation of the frequency bands is needed. One such mechanism is the **SPADE** system used on some early Intelsat satellites. Each SPADE transponder was divided into 794 simplex (64-kbps) PCM voice channels, along with a 128-kbps common signaling channel. The PCM channels were used in pairs to provide full duplex service. The total transponder bandwidth used was 50 Mbps for the uplink portion and another 50 Mbps for the downlink.

The common signaling channel was divided into units of 50 msec. A unit contained 50 slots of 1 msec (128 bits). Each slot was "owned" by one of (not more than) 50 ground stations. When a ground station had data to send, it picked a currently unused channel at random and wrote the number of that channel in its next 128-bit slot. If the selected channel was still unused when the request was seen on the downlink, the channel was considered allocated and all other stations refrained from trying to acquire it. If two or more stations tried to allocate the same channel in the same frame, a collision occurred and they had to try again later. When a station was finished using its channel, it sent a deallocation message in its slot on the common channel.

4.6.4. TDM

Like FDM, TDM is well understood and widely used in practice. It requires time synchronization for the slots, but this can be provided by a reference station, as described for slotted ALOHA above. Similarly to FDM, for a small and unvarying number of stations, the slot assignment can be set up in advance and never changed, but for a varying number of stations, or a fixed number of stations with time-varying loads, time slots must be assigned dynamically.

Slot assignment can be done in a centralized or a decentralized way. As an example of centralized slot assignment, let us consider the experimental **ACTS** (**Advanced Communication Technology Satellite**), which was designed for a few dozen stations (Palmer and White, 1990). ACTS was launched in 1992 and has four independent 110-Mbps TDM channels, two uplink and two downlink. Each channel is organized as a sequence of 1-msec frames, each frame containing 1728 time slots. Each time slot has a 64-bit payload, allowing each one to hold a 64-kbps voice channel.

The beams can be switched from one geographical area to another, but since moving the beam takes several slot times, channels originating or terminating in the same geographic area are normally assigned to contiguous time slots to increase dwell time and minimize time lost to beam motion. Thus time slot management requires a thorough knowledge of station geography to minimize the number of wasted time slots. For this and other reasons, time slot management is done by one of the ground stations, the **MCS** (**Master Control Station**).

The basic operation of ACTS is a continuous three-step process, each step taking 1 msec. In step 1, the satellite receives a frame and stores it in a 1728-entry onboard RAM. In step 2, an onboard computer copies each input entry to the corresponding output entry (possibly for the other antenna). In step 3, the output frame is transmitted on the downlink.

Initially, each station is assigned at least one time slot. To acquire additional channels (for new voice calls), a station sends a short request message to the MCS. Similarly, it can release an existing channel with a message to the MCS. These messages make use of a small number of overhead bits and provide a special control channel to the MCS with a capacity of about 13 messages/sec per station. The channels are dedicated; there is no contention for them.

Dynamic TDM slot allocation is also possible. Below we will discuss three schemes. In each of these, TDM frames are divided into time slots, with each slot having a (temporary) owner. Only the owner may use a time slot.

The first scheme assumes that there are more slots than stations, so each station can be assigned a home slot (Binder, 1975). If there are more slots than stations, the extra slots are not assigned to anyone. If the owner of a slot does not want it during the current group, it goes idle. An empty slot is a signal to everyone else that the owner has no traffic. During the next frame, the slot becomes available to anyone who wants it, on a contention (ALOHA) basis.

If the owner wants to retrieve "his" home slot, he transmits a frame, thus forcing a collision (if there was other traffic). After a collision, everyone except the owner must desist from using the slot in the next frame. Thus the owner can always begin transmitting within two frame times in the worst case. At low channel utilization the system does not perform as well as normal slotted ALOHA, since after each collision, the collidees must abstain for one frame to see if the owner wants the slot back. Fig. 4-51(a) shows a frame with eight slots, seven of which are owned by *G*, *A*, *F*, *E*, *B*, *C*, and *D*, respectively. The eighth slot is not owned by anyone and can be fought over.

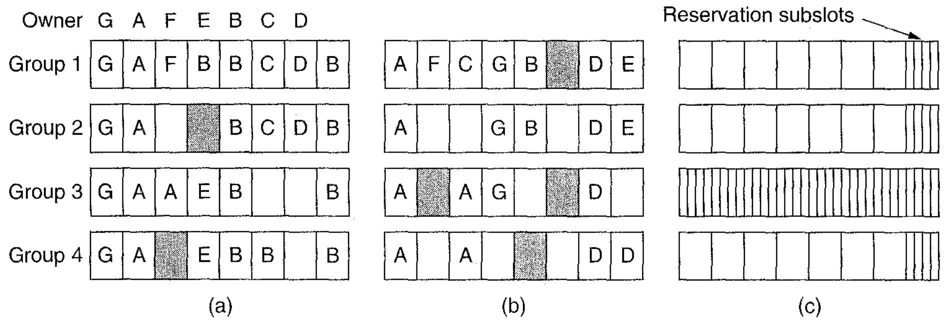


Fig. 4-51. Reservation schemes. (a) Binder. (b) Crowther. (c) Roberts. The shaded boxes indicate collisions. For each of the three schemes, four consecutive groups of slots are shown.

A second scheme is applicable even when the number of stations is unknown and varying (Crowther et al., 1973). In this method, slots do not have permanent owners, as in Binder's. Instead, stations compete for slots using slotted ALOHA. Whenever a transmission is successful, the station making the successful transmission is entitled to that slot in the next frame as well. Thus, as long as a station has data to send, it can continue doing so indefinitely (subject to some "Please-do-not-be-a-pig'' rules). In essence the proposal allows a dynamic mix of slotted ALOHA and TDM, with the number of slots devoted to each varying with demand. Fig. 4-51(b) also shows a frame with eight slots. Initially, *E* is using the last slot, but after two frames, it no longer needs it. It lies idle for one frame, and then *D* picks it up and keeps it until it is done.

A third scheme, due to Roberts (1973), requires stations to make advance requests before transmitting. Each frame contains, say, one special slot [the last one in Fig. 4-51(c)] which is divided into *V* smaller subslots used to make reservations. When a station wants to send data, it broadcasts a short request frame in a randomly-chosen reservation subslot. If the reservation is successful (i.e., no collision), then the next regular slot (or slots) is reserved. At all times everyone must keep track of the queue length (number of slots reserved), so that when any station makes a successful reservation it will know how many data slots to skip before transmitting. Stations need not keep track of *who* is queued up; they merely need to know how long the queue is. When the queue length drops to zero, all slots revert to reservation subslots, to speed up the reservation process.

Although TDM is widely used, both with and without reservation schemes, it, too, has some shortcomings. For one, it requires all stations to synchronize in time, which is not entirely trivial in practice because satellites tend to drift in orbit, which changes the propagation time to each ground station. It also requires each ground station to be capable of extremely high burst speeds. For example, even though an ACTS station may have only one 64-kbps channel, it must be capable of putting out a 64-bit burst in a 578-nsec time slot. In other words, it must actually operate at 110 Mbps. In contrast, a 64-kbps FDM station really operates at 64 kbps.

4.6.5. CDMA

The final scheme is CDMA. CDMA avoids the time synchronization problem and also the channel allocation problem. It is completely decentralized and fully dynamic.

However, it has three main disadvantages. First, the capacity of a CDMA channel in the presence of noise and uncoordinated stations is typically lower than what TDM can achieve. Second, with 128 chips/bit (a common value), although the bit rate may not be high, the chip rate will be, necessitating a fast (read: expensive) transmitter. Third, few practicing engineers actually understand CDMA, which generally does not increase the chances of their using it, even if it is the best method for a particular application. Nevertheless, CDMA has been used by the military for decades and is now becoming more common in commercial applications as well.

4.7. SUMMARY

Some networks have a single channel that is used for all communication. In these networks, the key design issue is the allocation of this channel among the competing stations wishing to use it. Numerous channel allocation algorithms have been devised. A summary of some of the more important channel allocation methods is given in Fig. 4-52.

The simplest allocation schemes are FDM and TDM. These are efficient when the number of stations is small and the traffic is continuous. Both are widely used under these circumstances, for example, for dividing up the bandwidth in satellite links used as telephone trunks.

When the number of stations is large and variable or the traffic bursty, FDM and TDM are poor choices. The ALOHA protocol, with and without slotting and control, has been proposed as an alternative. ALOHA and its many variants and derivatives have been widely discussed, analyzed, and used in real systems.

When the state of the channel can be sensed, stations can avoid starting a transmission while another station is transmitting. This technique, carrier sensing, has led to a variety of protocols that can be used on LANs and MANs.

A class of protocols that eliminate contention altogether, or at least reduce it considerably, is known. Binary countdown completely eliminates contention.

Method	Description
FDM	Dedicate a frequency band to each station
TDM	Dedicate a time slot to each station
Pure ALOHA	Unsynchronized transmission at any instant
Slotted ALOHA	Random transmission in well-defined time slots
1-persistent CSMA	Standard carrier sense multiple access
Nonpersistent CSMA	Random delay when channel is sensed busy
P-persistent CSMA	CSMA, but with a probability of p of persisting
CSMA/CD	CSMA, but abort on detecting a collision
Bit map	Round robin scheduling using a bit map
Binary countdown	Highest numbered ready station goes next
Tree walk	Reduced contention by selective enabling
Wavelength division	A dynamic FDM scheme for fiber
MACA, MACAW	Wireless LAN protocols
GSM	FDM plus TDM for cellular radio
CDPD	Packet radio within an AMPS channel
CDMA	Everybody speak at once but in a different language
Ethernet	CSMA/CD with binary exponential backoff
Token bus	Logical ring on a physical bus
Token ring	Capture the token to send a frame
DQDB	Distributed queuing on a two-bus MAN
FDDI	Fiber-optic token ring
HIPPI	Crossbar using 50-100 twisted pairs
Fibre channel	Crossbar using fiber optics
SPADE	FDM with dynamic channel allocation
ACTS	TDM with centralized slot allocation
Binder	TDM with ALOHA when slot owner is not interested
Crowther	ALOHA with slot owner getting to keep it
Roberts	Channel time reserved in advance by ALOHA

Fig. 4-52. Channel allocation methods and systems for a common channel.

The tree walk protocol reduces it by dynamically dividing the stations into two disjoint groups, one of which is permitted to transmit and one of which is not. It tries to make the division in such a way that only one station that is ready to send is permitted to do so.

Wireless LANs have their own problems and solutions. The biggest problem is caused by hidden stations, so CSMA does not work. One class of solutions, typified by MACA, attempts to stimulate transmissions around the destination, to make CSMA work better.

For mobile computers and telephones, cellular radio is the up-and-coming technology. GSM, CDPD, and CDMA are widely used.

The IEEE 802 LANs are: CSMA/CD, token bus, and token ring. Each of these has its own unique advantages and disadvantages, and each has found its own user community and will probably continue to serve that community for years to come. Convergence to a single LAN standard is an unlikely event. A new addition to this family is DQDB, being sold as a MAN in many cities.

An organization with multiple LANs often connects them with bridges. When a bridge connects two or more different kinds of LANs, new problems arise, some of them insoluble.

While the 802 LANs are the work horses of the day, the race horses are FDDI, fast Ethernet, HIPPI, and fibre channel. All of these offer bandwidth in the 100 Mbps range and up.

Finally, satellite networks also use multiple access channels (for the uplink). Various channel allocation methods are used here, including ALOHA, FDM, TDM, and CDMA.

PROBLEMS

1. A group of N stations share a 56-kbps pure ALOHA channel. Each station outputs a 1000-bit frame on an average of once every 100 sec, even if the previous one has not yet been sent (e.g., the stations are buffered). What is the maximum value of N?

2. Consider the delay of pure ALOHA versus slotted ALOHA at low load. Which one is less? Explain your answer.

3. Ten thousand airline reservation stations are competing for the use of a single slotted ALOHA channel. The average station makes 18 requests/hour. A slot is 125 μsec. What is the approximate total channel load?

4. A large population of ALOHA users manages to generate 50 requests/sec, including both originals and retransmissions. Time is slotted in units of 40 msec.
(a) What is the chance of success on the first attempt?
(b) What is the probability of exactly k collisions and then a success?
(c) What is the expected number of transmission attempts needed?

5. Measurements of a slotted ALOHA channel with an infinite number of users show that 10 percent of the slots are idle.
(a) What is the channel load, G?
(b) What is the throughput?
(c) Is the channel underloaded or overloaded?

6. In an infinite-population slotted ALOHA system, the mean number of slots a station waits between a collision and its retransmission is 4. Plot the delay versus throughput curve for this system.

7. A LAN uses Mok and Ward's version of binary countdown. At a certain instant, the ten stations have the virtual station numbers 8, 2, 4, 5, 1, 7, 3, 6, 9, and 0. The next three stations to send are 4, 3, and 9, in that order. What are the new virtual station numbers after all three have finished their transmissions?

8. Sixteen stations are contending for the use of a shared channel using the adaptive tree walk protocol. If all the stations whose addresses are prime numbers suddenly become ready at once, how many bit slots are needed to resolve the contention?

9. A collection of 2^n stations uses the adaptive tree walk protocol to arbitrate access to a shared cable. At a certain instant two of them become ready. What are the minimum, maximum, and mean number of slots to walk the tree if $2^n \gg 1$?

10. The wireless LANs that we studied used protocols such as MACA instead of CSMA/CD. Under what conditions would it be possible to use CSMA/CD instead?

11. What properties do the WDMA and GSM channel access protocols have in common?

12. Using the GSM framing structure as given in Fig. 4-14, determine how often any given user may send a data frame.

13. Suppose that A, B, and C are simultaneously transmitting 0 bits using a CDMA system with the chip sequences of Fig. 4-16(b). What is the resulting chip sequence?

14. In the discussion about orthogonality of CDMA chip sequences, it was stated that if $\mathbf{S} \bullet \mathbf{T} = 0$ then $\mathbf{S} \bullet \overline{\mathbf{T}}$ is also 0. Prove this.

15. Consider a different way of looking at the orthogonality property of CDMA chip sequences. Each bit in a pair of sequence can match or not match. Express the orthogonality property in terms of matches and mismatches.

16. A CDMA receiver gets the following chips: (−1 +1 −3 +1 −1 −3 +1 +1). Assuming the chip sequences defined in Fig. 4-16(b), which stations transmitted, and which bits did each one send?

17. A seven-story office building has 15 adjacent offices per floor. Each office contains a wall socket for a terminal in the front wall, so the sockets form a rectangular grid in the vertical plane, with a separation of 4 m between sockets, both horizontally and vertically. Assuming that it is feasible to run a straight cable between any pair of sockets, horizontally, vertically, or diagonally, how many meters of cable are needed to connect all sockets using
(a) a star configuration with a single router in middle?
(b) an 802.3 LAN?
(c) a ring net (without a wire center)?

18. What is the baud rate of the standard 10-Mbps 802.3 LAN?

19. A 1-km-long, 10-Mbps CSMA/CD LAN (not 802.3) has a propagation speed of 200 m/μsec. Data frames are 256 bits long, including 32 bits of header, checksum, and other overhead. The first bit slot after a successful transmission is reserved for the receiver to capture the channel to send a 32-bit acknowledgement frame. What is the effective data rate, excluding overhead, assuming that there are no collisions?

20. Two CSMA/CD stations are each trying to transmit long (multiframe) files. After each frame is sent, they contend for the channel using the binary exponential backoff algorithm. What is the probability that the contention ends on round k, and what is the mean number of rounds per contention period?

21. Consider building a CSMA/CD network running at 1 Gbps over a 1-km cable with no repeaters. The signal speed in the cable is 200,000 km/sec. What is the minimum frame size?

22. Sketch the Manchester encoding for the bit stream: 0001110101.

23. Sketch the differential Manchester encoding for the bit stream of the previous problem. Assume the line is initially in the low state.

24. A token bus system works like this. When the token arrives at a station, a timer is reset to 0. The station then begins transmitting priority 6 frames until the timer reaches *T6*. Then it switches over to priority 4 frames until the timer reaches *T4*. This algorithm is then repeated with priority 2 and priority 0. If all stations have timer values of 40, 80, 90, and 100 msec for *T6* through *T0*, respectively, what fraction of the total bandwidth is reserved for each priority class?

25. What happens in a token bus if a station accepts the token and then crashes immediately? How does the protocol described in the text handle this case?

26. At a transmission rate of 5 Mbps and a propagation speed of 200 m/μsec, to how many meters of cable is the 1-bit delay in a token ring interface equivalent?

27. The delay around a token ring must be enough to contain the entire token. If the wire is not long enough, some artificial delay must be introduced. Explain why this extra delay is necessary in the content of a 24-bit token and a ring with only 16 bits of delay.

28. A very heavily loaded 1-km-long, 10-Mbps token ring has a propagation speed of 200 m/μsec. Fifty stations are uniformly spaced around the ring. Data frames are 256 bits, including 32 bits of overhead. Acknowledgements are piggybacked onto the data frames and are thus included as spare bits within the data frames and are effectively free. The token is 8 bits. Is the effective data rate of this ring higher or lower than the effective data rate of a 10-Mbps CSMA/CD network?

29. In a token ring the sender removes the frame. What modifications to the system would be needed to have the receiver remove the frame instead, and what would the consequences be?

30. A 4-Mbps token ring has a token-holding timer value of 10 msec. What is the longest frame that can be sent on this ring?

31. Does the use of a wire center have any influence on the performance of a token ring?

32. A fiber optic token ring used as a MAN is 200 km long and runs at 100 Mbps. After sending a frame, a station drains the frame from the ring before regenerating the token. The signal propagation speed in the fiber is 200,000 km/sec and the maximum frame size is 1K bytes. What is the maximum efficiency of the ring (ignoring all other sources of overhead)?

33. In Fig. 4-32, station *D* wants to send a cell. To which station does it want to send it?

34. The system of Fig. 4-32 is idle. A little later, stations *C*, *A*, and *B* become ready to send, in that order and in rapid succession. Assuming that no data frames are transmitted until all three have sent a request upstream, show the *RC* and *CD* values after each request and after the three data frames.

35. Ethernet is sometimes said to be inappropriate for real-time computing because the worst case retransmission interval is not bounded. Under what circumstances can the same argument be leveled at the token ring? Under what circumstances does the token ring have a known worst case? Assume the number of stations on the token ring is fixed and known.

36. Ethernet frames must be at least 64 bytes long to ensure that the transmitter is still going in the event of a collision at the far end of the cable. Fast Ethernet has the same 64 byte minimum frame size, but can get the bits out ten times faster. How is it possible to maintain the same minimum frame size?

37. Imagine two LAN bridges, both connecting a pair of 802.4 networks. The first bridge is faced with 1000 512-byte frames per second that must be forwarded. The second is faced with 200 4096-byte frames per second. Which bridge do you think will need the faster CPU? Discuss.

38. Suppose that the two bridges of the previous problem each connected an 802.4 LAN to an 802.5 LAN. Would that change have any influence on the previous answer?

39. A bridge between an 802.3 LAN and an 802.4 LAN has a problem with intermittent memory errors. Can this problem cause undetected errors with transmitted frames, or will these all be caught by the frame checksums?

40. A university computer science department has 3 Ethernet segments, connected by two transparent bridges into a linear network. One day the network administrator quits and is hastily replaced by someone from the computer center, which is an IBM token ring shop. The new administrator, noticing that the ends of the network are not connected, quickly orders a new transparent bridge and connects both loose ends to it, making a closed ring. What happens next?

41. A large FDDI ring has 100 stations and a token rotation time of 40 msec. The token-holding time is 10 msec. What is the maximum achievable efficiency of the ring?

42. Consider building a supercomputer interconnect using the HIPPI approach, but modern technology. The data path is now 64 bits wide, and a word can be sent every 10 nsec. What is the bandwidth of the channel?

43. In the text it was stated that a satellite with two uplink and one downlink slotted ALOHA channels can achieve a downlink utilization of 0.736, given an infinite amount of buffer space. Show how this result can be obtained.

5

THE NETWORK LAYER

The network layer is concerned with getting packets from the source all the way to the destination. Getting to the destination may require making many hops at intermediate routers along the way. This function clearly contrasts with that of the data link layer, which has the more modest goal of just moving frames from one end of a wire to the other. Thus the network layer is the lowest layer that deals with end-to-end transmission. For more information about it, see (Huitema, 1995; and Perlman, 1992).

To achieve its goals, the network layer must know about the topology of the communication subnet (i.e., the set of all routers) and choose appropriate paths through it. It must also take care to choose routes to avoid overloading some of the communication lines and routers while leaving others idle. Finally, when the source and destination are in different networks, it is up to the network layer to deal with these differences and solve the problems that result from them. In this chapter we will study all these issues and illustrate them with our two running examples, the Internet and ATM.

5.1. NETWORK LAYER DESIGN ISSUES

In the following sections we will provide an introduction to some of the issues that the designers of the network layer must grapple with. These issues include the service provided to the transport layer and the internal design of the subnet.

5.1.1. Services Provided to the Transport Layer

The network layer provides services to the transport layer at the network layer/transport layer interface. This interface is often especially important for another reason: it frequently is the interface between the carrier and the customer, that is, the boundary of the subnet. The carrier often has control of the protocols and interfaces up to and including the network layer. Its job is to deliver packets given to it by its customers. For this reason, this interface must be especially well defined.

The network layer services have been designed with the following goals in mind.

1. The services should be independent of the subnet technology.
2. The transport layer should be shielded from the number, type, and topology of the subnets present.
3. The network addresses made available to the transport layer should use a uniform numbering plan, even across LANs and WANs.

Given these goals, the designers of the network layer have a lot of freedom in writing detailed specifications of the services to be offered to the transport layer. This freedom often degenerates into a raging battle between two warring factions. The discussion centers on the question of whether the network layer should provide connection-oriented service or connectionless service.

One camp (represented by the Internet community) argues that the subnet's job is moving bits around and nothing else. In their view (based on nearly 30 years of actual experience with a real, working computer network), the subnet is inherently unreliable, no matter how it is designed. Therefore, the hosts should accept the fact that it is unreliable and do error control (i.e., error detection and correction) and flow control themselves.

This viewpoint leads quickly to the conclusion that the network service should be connectionless, with primitives SEND PACKET and RECEIVE PACKET, and little else. In particular, no packet ordering and flow control should be done, because the hosts are going to do that anyway, and there is probably little to be gained by doing it twice. Furthermore, each packet must carry the full destination address, because each packet sent is carried independently of its predecessors, if any.

The other camp (represented by the telephone companies) argues that the subnet should provide a (reasonably) reliable, connection-oriented service. They claim 100 years of successful experience with the worldwide telephone system is a good guide. In this view, connections should have the following properties:

1. Before sending data, a network layer process on the sending side must set up a connection to its peer on the receiving side. This connection, which is given a special identifier, is then used until all the data have been sent, at which time it is explicitly released.

2. When a connection is set up, the two processes can enter into a negotiation about the parameters, quality, and cost of the service to be provided.

3. Communication is in both directions, and packets are delivered in sequence.

4. Flow control is provided automatically to prevent a fast sender from dumping packets into the pipe at a higher rate than the receiver can take them out, thus leading to overflow.

Other properties, such as guaranteed delivery, explicit confirmation of delivery, and high priority packets are optional. As we pointed out in Chap. 1, connectionless service is like the postal system, and connection-oriented service is like the telephone system.

The argument between connection-oriented and connectionless service really has to do with where to put the complexity. In the connection-oriented service, it is in the network layer (subnet); in the connectionless service, it is in the transport layer (hosts). Supporters of connectionless service say that user computing power has become cheap, so that there is no reason not to put the complexity in the hosts. Furthermore, they argue that the subnet is a major (inter)national investment that will last for decades, so it should not be cluttered up with features that may become obsolete quickly but will have to be calculated into the price structure for many years. Furthermore, some applications, such as digitized voice and real-time data collection may regard *speedy* delivery as much more important than *accurate* delivery.

On the other hand, supporters of connection-oriented service say that most users are not interested in running complex transport layer protocols in their machines. What they want is reliable, trouble-free service, and this service can be best provided with network layer connections. Furthermore, some services, such as real time audio and video are much easier to provide on top of a connection-oriented network layer than on top of a connectionless network layer.

Although it is rarely discussed in these terms, two separate issues are involved here. First, whether the network is connection-oriented (setup required) or connectionless (no setup required). Second, whether it is reliable (no lost, duplicated, or garbled packets) or unreliable (packets can be lost, duplicated, or garbled). In theory, all four combinations exist, but the dominant combinations are reliable connection-oriented and unreliable connectionless, so the other two tend to get lost in the noise.

These two camps are represented by our two running examples. The Internet has a connectionless network layer, and ATM networks have a connection-oriented network layer. An obvious question arises about how the Internet works when it runs over an ATM-based, carrier-provided subnet. The answer is that the source host first establishes an ATM network layer connection to the destination host and then sends independent (IP) packets over it, as shown in Fig. 5-1. Although this approach works, it is inefficient because certain functionality is in both layers. For example, the ATM network layer guarantees that packets are always delivered in order, but the TCP code still contains the full mechanism for managing and reordering out-of-order packets. For more information about how to run IP over ATM, see RFC 1577 and (Armitage and Adams, 1995).

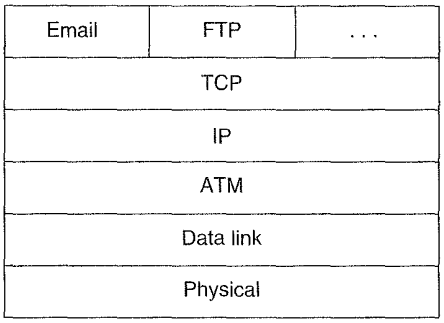


Fig. 5-1. Running TCP/IP over an ATM subnet.

5.1.2. Internal Organization of the Network Layer

Having looked at the two classes of service the network layer can provide to its users, it is time to see how it works inside. There are basically two different philosophies for organizing the subnet, one using connections and the other working connectionless. In the context of the *internal* operation of the subnet, a connection is usually called a **virtual circuit**, in analogy with the physical circuits set up by the telephone system. The independent packets of the connectionless organization are called **datagrams**, in analogy with telegrams.

Virtual circuits are generally used in subnets whose primary service is connection-oriented, so we will describe them in that context. The idea behind virtual circuits is to avoid having to choose a new route for every packet or cell sent. Instead, when a connection is established, a route from the source machine to the destination machine is chosen as part of the connection setup and remembered. That route is used for all traffic flowing over the connection, exactly the same way that the telephone system works. When the connection is released, the virtual circuit is also terminated.

In contrast, with a datagram subnet no routes are worked out in advance, even if the service is connection-oriented. Each packet sent is routed independently of its predecessors. Successive packets may follow different routes. While datagram subnets have to do more work, they are also generally more robust and adapt to failures and congestion more easily than virtual circuit subnets. We will discuss the pros and cons of the two approaches later.

If packets flowing over a given virtual circuit always take the same route through the subnet, each router must remember where to forward packets for each of the currently open virtual circuits passing through it. Every router must maintain a table with one entry per open virtual circuit passing through it. Each packet traveling through the subnet must contain a virtual circuit number field in its header, in addition to sequence numbers, checksums, and the like. When a packet arrives at a router, the router knows on which line it arrived and what the virtual circuit number is. Based on only this information, the packet must be forwarded on the correct output line.

When a network connection is set up, a virtual circuit number not already in use on that machine is chosen as the connection identifier. Since each machine chooses virtual circuit numbers independently, these numbers have only local significance. If they were globally significant over the whole network, it is likely that two virtual circuits bearing the same global virtual circuit number might pass through some intermediate router, leading to ambiguities.

Because virtual circuits can be initiated from either end, a problem occurs when call setups are propagating in both directions at once along a chain of routers. At some point they have arrived at adjacent routers. Each router must now pick a virtual circuit number to use for the (full-duplex) circuit it is trying to establish. If they have been programmed to choose the lowest number not already in use on the link, they will pick the same number, causing two unrelated virtual circuits over the same physical line to have the same number. When a data packet arrives later, the receiving router has no way of telling whether it is a forward packet on one circuit or a reverse packet on the other. If circuits are simplex, there is no ambiguity.

Note that every process must be required to indicate when it is through using a virtual circuit, so that the virtual circuit can be purged from the router tables to recover the space. In public networks, the motivation is the stick rather than the carrot: users are invariably charged for connect time as well as for data transported. In addition, some provision must be made for dealing with machines that terminate their virtual circuits by crashing rather than politely releasing them when done.

So much for the use of virtual circuits internal to the subnet. The other possibility is to use datagrams internally, in which case the routers do not have a table with one entry for each open virtual circuit. Instead, they have a table telling which outgoing line to use for each possible destination router. These tables are also needed when virtual circuits are used internally, to determine the route for a setup packet.

Each datagram must contain the full destination address. For a large network, these addresses can be quite long (e.g., a dozen bytes or more). When a packet comes in, the router looks up the outgoing line to use and sends the packet on its way. Also, the establishment and release of network or transport layer connections do not require any special work on the part of the routers.

5.1.3. Comparison of Virtual Circuit and Datagram Subnets

Both virtual circuits and datagrams have their supporters and their detractors. We will now attempt to summarize the arguments both ways. The major issues are listed in Fig. 5-2, although purists could probably find a counterexample for everything in the figure.

Issue	Datagram subnet	VC subnet
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Subnet does not hold state information	Each VC requires subnet table space
Routing	Each packet is routed independently	Route chosen when VC is set up; all packets follow this route
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Congestion control	Difficult	Easy if enough buffers can be allocated in advance for each VC

Fig. 5-2. Comparison of datagram and virtual circuit subnets.

Inside the subnet, several trade-offs exist between virtual circuits and datagrams. One trade-off is between router memory space and bandwidth. Virtual circuits allow packets to contain circuit numbers instead of full destination addresses. If the packets tend to be fairly short, a full destination address in every packet may represent a significant amount of overhead, and hence wasted bandwidth. The price paid for using virtual circuits internally is the table space within the routers. Depending upon the relative cost of communication circuits versus router memory, one or the other may be cheaper.

Another trade-off is setup time versus address parsing time. Using virtual circuits requires a setup phase, which takes time and consumes resources. However, figuring out what to do with a data packet in a virtual circuit subnet is easy: the router just uses the circuit number to index into a table to find out where the packet goes. In a datagram subnet, a more complicated procedure is required to determine where the packet goes.

Virtual circuits have some advantages in avoiding congestion within the subnet because resources can be reserved in advance, when the connection is established. Once the packets start arriving, the necessary bandwidth and router capacity will be there. With a datagram subnet, congestion avoidance is more difficult.

For transaction processing systems (e.g., stores calling up to verify credit card purchases), the overhead required to set up and clear a virtual circuit may easily dwarf the use of the circuit. If the majority of the traffic is expected to be of this kind, the use of switched virtual circuits inside the subnet makes little sense. On the other hand, permanent virtual circuits, which are set up manually and last for months or years, may be useful here.

Virtual circuits also have a vulnerability problem. If a router crashes and loses its memory, even if it comes back up a second later, all the virtual circuits passing through it will have to be aborted. In contrast, if a datagram router goes down, only those users whose packets were queued up in the router at the time will suffer, and maybe not even all those, depending upon whether they have already been acknowledged or not. The loss of a communication line is fatal to virtual circuits using it but can be easily compensated for if datagrams are used. Datagrams also allow the routers to balance the traffic throughout the subnet, since routes can be changed halfway through a connection.

It is worth explicitly pointing out that the service offered (connection-oriented or connectionless) is a separate issue from the subnet structure (virtual circuit or datagram). In theory, all four combinations are possible. Obviously, a virtual circuit implementation of a connection-oriented service and a datagram implementation of a connectionless service are reasonable. Implementing connections using datagrams also makes sense when the subnet is trying to provide a highly robust service.

The fourth possibility, a connectionless service on top of a virtual circuit subnet, seems strange but certainly occurs. The obvious example is running IP over an ATM subnet. Here it is desired to run an existing connectionless protocol over a new connection-oriented network layer. As mentioned earlier, this is more of an ad hoc solution to a problem than a good design. In a new system designed to run over an ATM subnet, one would not normally put a connectionless protocol like IP over a connection-oriented network layer like ATM and then layer a connection-oriented transport protocol on top of the connectionless protocol. Examples of all four cases are shown in Fig. 5-3.

5.2. ROUTING ALGORITHMS

The main function of the network layer is routing packets from the source machine to the destination machine. In most subnets, packets will require multiple hops to make the journey. The only notable exception is for broadcast

Upper layer	Type of subnet	
	Datagram	Virtual circuit
Connectionless	UDP over IP	UDP over IP over ATM
Connection-oriented	TCP over IP	ATM AAL1 over ATM

Fig. 5-3. Examples of different combinations of service and subnet structure.

networks, but even here routing is an issue if the source and destination are not on the same network. The algorithms that choose the routes and the data structures that they use are a major area of network layer design.

The **routing algorithm** is that part of the network layer software responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagrams internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new virtual circuit is being set up. Thereafter, data packets just follow the previously established route. The latter case is sometimes called **session routing**, because a route remains in force for an entire user session (e.g., a login session at a terminal or a file transfer).

Regardless of whether routes are chosen independently for each packet or only when new connections are established, there are certain properties that are desirable in a routing algorithm: correctness, simplicity, robustness, stability, fairness, and optimality. Correctness and simplicity hardly require comment, but the need for robustness may be less obvious at first. Once a major network comes on the air, it may be expected to run continuously for years without systemwide failures. During that period there will be hardware and software failures of all kinds. Hosts, routers, and lines will go up and down repeatedly, and the topology will change many times. The routing algorithm should be able to cope with changes in the topology and traffic without requiring all jobs in all hosts to be aborted and the network to be rebooted every time some router crashes.

Stability is also an important goal for the routing algorithm. There exist routing algorithms that never converge to equilibrium, no matter how long they run. Fairness and optimality may sound obvious—surely no one would oppose them—but as it turns out, they are often contradictory goals. As a simple example of this conflict, look at Fig. 5-4. Suppose that there is enough traffic between A and A', between B and B', and between C and C' to saturate the horizontal links. To maximize the total flow, the X to X' traffic should be shut off altogether.

Unfortunately, X and X' may not see it that way. Evidently, some compromise between global efficiency and fairness to individual connections is needed.

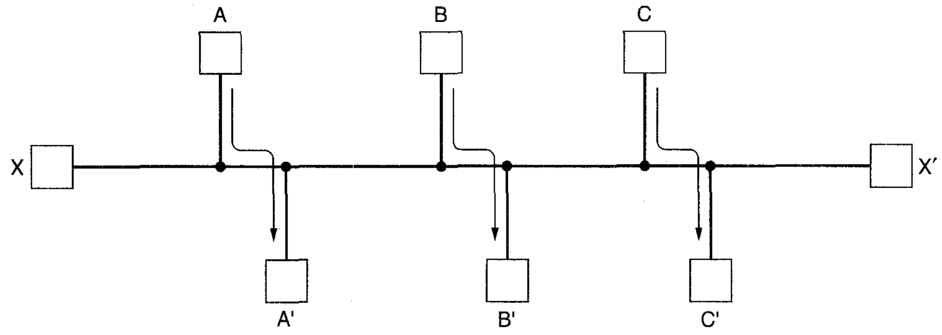


Fig. 5-4. Conflict between fairness and optimality.

Before we can even attempt to find trade-offs between fairness and optimality, we must decide what it is we seek to optimize. Minimizing mean packet delay is an obvious candidate, but so is maximizing total network throughput. Furthermore, these two goals are also in conflict, since operating any queueing system near capacity implies a long queueing delay. As a compromise, many networks attempt to minimize the number of hops a packet must make, because reducing the number of hops tends to improve the delay and also reduce the amount of bandwidth consumed, which tends to improve the throughput as well.

Routing algorithms can be grouped into two major classes: nonadaptive and adaptive. **Nonadaptive algorithms** do not base their routing decisions on measurements or estimates of the current traffic and topology. Instead, the choice of the route to use to get from I to J (for all I and J) is computed in advance, off-line, and downloaded to the routers when the network is booted. This procedure is sometimes called **static routing**.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get their information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every ΔT sec, when the load changes, or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time). In the following sections we will discuss a variety of routing algorithms, both static and dynamic.

5.2.1. The Optimality Principle

Before getting into specific algorithms, it may be helpful to note that one can make a general statement about optimal routes without regard to network topology or traffic. This statement is known as the **optimality principle**. It states that if router J is on the optimal path from router I to router K, then the optimal path from *J* to *K* also falls along the same route. To see this, call the part of the route from *I* to *J* r_1 and the rest of the route r_2. If a route better than r_2 existed from *J* to *K*, it could be concatenated with r_1 to improve the route from *I* to *K*, contradicting our statement that $r_1 r_2$ is optimal.

As a direct consequence of the optimality principle, we can see that the set of optimal routes from all sources to a given destination form a tree rooted at the destination. Such a tree is called a **sink tree** and is illustrated in Fig. 5-5 where the distance metric is the number of hops. Note that a sink tree is not necessarily unique; other trees with the same path lengths may exist. The goal of all routing algorithms is to discover and use the sink trees for all routers.

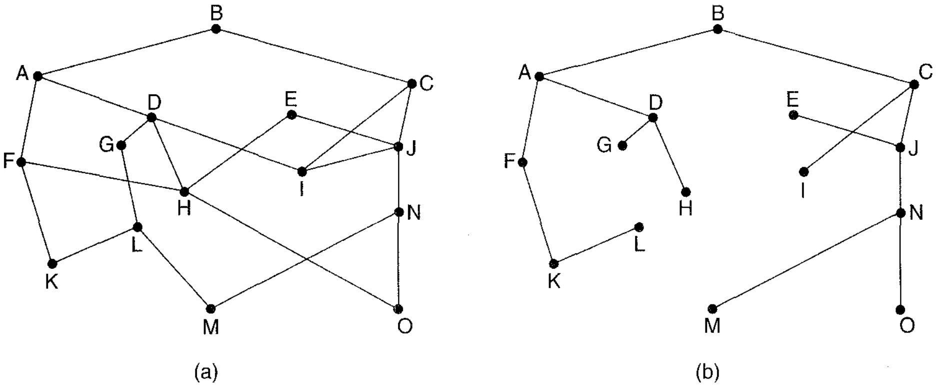


Fig. 5-5. (a) A subnet. (b) A sink tree for router *B*.

Since a sink tree is indeed a tree, it does not contain any loops, so each packet will be delivered within a finite and bounded number of hops. In practice, life is not quite this easy. Links and routers can go down and come back up during operation, so different routers may have different ideas about the current topology. Also, we have quietly finessed the issue of whether each router has to individually acquire the information on which to base its sink tree computation, or whether this information is collected by some other means. We will come back to these issues shortly. Nevertheless, the optimality principle and the sink tree provide a benchmark against which other routing algorithms can be measured.

In the next three sections, we will look at three different static routing algorithms. After that we will move on to adaptive ones.

5.2.2. Shortest Path Routing

Let us begin our study of routing algorithms with a technique that is widely used in many forms because it is simple and easy to understand. The idea is to build a graph of the subnet, with each node of the graph representing a router and each arc of the graph representing a communication line (often called a link). To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.

The concept of a **shortest path** deserves some explanation. One way of measuring path length is the number of hops. Using this metric, the paths *ABC* and *ABE* in Fig. 5-6 are equally long. Another metric is the geographic distance in kilometers, in which case *ABC* is clearly much longer than *ABE* (assuming the figure is drawn to scale).

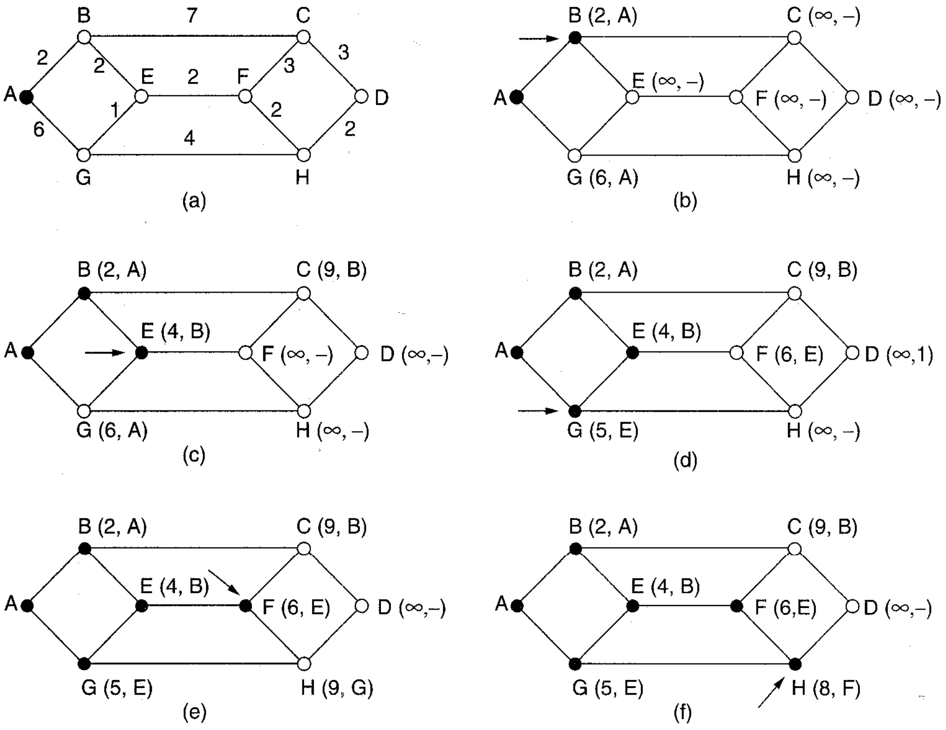


Fig. 5-6. The first five steps used in computing the shortest path from *A* to *D*. The arrows indicate the working node.

However, many other metrics are also possible besides hops and physical distance. For example, each arc could be labeled with the mean queueing and transmission delay for some standard test packet as determined by hourly test runs. With this graph labeling, the shortest path is the fastest path, rather than the path with the fewest arcs or kilometers.

In the most general case, the labels on the arcs could be computed as a function of the distance, bandwidth, average traffic, communication cost, mean queue length, measured delay, and other factors. By changing the weighting function, the algorithm would then compute the "shortest" path measured according to any one of a number of criteria, or a combination of criteria.

Several algorithms for computing the shortest path between two nodes of a graph are known. This one is due to Dijkstra (1959). Each node is labeled (in parentheses) with its distance from the source node along the best known path. Initially, no paths are known, so all nodes are labeled with infinity. As the algorithm proceeds and paths are found, the labels may change, reflecting better paths. A label may be either tentative or permanent. Initially, all labels are tentative. When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent and never changed thereafter.

To illustrate how the labeling algorithm works, look at the weighted, undirected graph of Fig. 5-6(a), where the weights represent, for example, distance. We want to find the shortest path from *A* to *D*. We start out by marking node *A* as permanent, indicated by a filled in circle. Then we examine, in turn, each of the nodes adjacent to *A* (the working node), relabeling each one with the distance to *A*. Whenever a node is relabeled, we also label it with the node from which the probe was made, so we can reconstruct the final path later. Having examined each of the nodes adjacent to *A*, we examine all the tentatively labeled nodes in the whole graph and make the one with the smallest label permanent, as shown in Fig. 5-6(b). This one becomes the new working node.

We now start at *B*, and examine all nodes adjacent to it. If the sum of the label on *B* and the distance from *B* to the node being considered is less than the label on that node, we have a shorter path, so the node is relabeled.

After all the nodes adjacent to the working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively labeled node with the smallest value. This node is made permanent and becomes the working node for the next round. Figure 5-6 shows the first five steps of the algorithm.

To see why the algorithm works, look at Fig. 5-6(c). At that point we have just made *E* permanent. Suppose that there were a shorter path than *ABE*, say *AXYZE*. There are two possibilities: either node *Z* has already been made permanent, or it has not been. If it has, then *E* has already been probed (on the round following the one when *Z* was made permanent), so the *AXYZE* path has not escaped our attention.

Now consider the case where *Z* is still tentatively labeled. Either the label at *Z* is greater than or equal to that at *E*, in which case *AXYZE* cannot be a shorter path than *ABE*, or it is less than that of *E*, in which case *Z* and not *E* will become permanent first, allowing *E* to be probed from *Z*.

This algorithm is given in Fig. 5-7. The only difference between the program and the algorithm described above is that in Fig. 5-7, we compute the shortest path starting at the terminal node, *t*, rather than at the source node, *s*. Since the shortest path from *t* to *s* in an undirected graph is the same as the shortest path from *s* to *t*, it does not matter at which end we begin (unless there are several shortest paths, in which case reversing the search might discover a different one). The reason for searching backward is that each node is labeled with its predecessor rather than its successor. When copying the final path into the output variable, *path*, the path is thus reversed. By reversing the search, the two effects cancel, and the answer is produced in the correct order.

5.2.3. Flooding

Another static algorithm is **flooding**, in which every incoming packet is sent out on every outgoing line except the one it arrived on. Flooding obviously generates vast numbers of duplicate packets, in fact, an infinite number unless some measures are taken to damp the process. One such measure is to have a hop counter contained in the header of each packet, which is decremented at each hop, with the packet being discarded when the counter reaches zero. Ideally, the hop counter should be initialized to the length of the path from source to destination. If the sender does not know how long the path is, it can initialize the counter to the worst case, namely, the full diameter of the subnet.

An alternative technique for damming the flood is to keep track of which packets have been flooded, to avoid sending them out a second time. One way to achieve this goal is to have the source router put a sequence number in each packet it receives from its hosts. Each router then needs a list per source router telling which sequence numbers originating at that source have already been seen. If an incoming packet is on the list, it is not flooded.

To prevent the list from growing without bound, each list should be augmented by a counter, *k*, meaning that all sequence numbers through *k* have been seen. When a packet comes in, it is easy to check if the packet is a duplicate; if so, it is discarded. Furthermore, the full list below *k* is not needed, since *k* effectively summarizes it.

A variation of flooding that is slightly more practical is **selective flooding**. In this algorithm the routers do not send every incoming packet out on every line, only on those lines that are going approximately in the right direction. There is usually little point in sending a westbound packet on an eastbound line unless the topology is extremely peculiar.

Flooding is not practical in most applications, but it does have some uses. For example, in military applications, where large numbers of routers may be blown to bits at any instant, the tremendous robustness of flooding is highly desirable. In distributed database applications, it is sometimes necessary to update all the databases concurrently, in which case flooding can be useful. A third possible use of flooding is as a metric against which other routing algorithms can be compared. Flooding always chooses the shortest path, because it chooses every possible path in parallel. Consequently, no other algorithm can produce a shorter delay (if we ignore the overhead generated by the flooding process itself).

```
#define MAX_NODES 1024                /* maximum number of nodes */
#define INFINITY 1000000000           /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES];/* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                      /* the path being worked on */
      int predecessor;                /* previous node */
      int length;                     /* length from source to this node */
      enum {permanent, tentative} label; /* label state */
  } state[MAX_NODES];

  int i, k, min;
  struct state *
               p;
  for (p = &state[0]; p < &state[n]; p++) { /* initialize state */
      p->predecessor = -1;
      p->length = INFINITY;
      p->label = tentative;
  }
  state[t].length = 0;  state[t].label = permanent;
  k = t;                              /* k is the initial working node */
  do {                                /* Is there a better path from k? */
      for (i = 0; i < n; i++)         /* this graph has n nodes */
          if (dist[k][i] != 0 && state[i].label == tentative) {
               if (state[k].length + dist[k][i] < state[i].length) {
                    state[i].predecessor = k;
                    state[i].length = state[k].length + dist[k][i];
               }
          }

      /* Find the tentatively labeled node with the smallest label. */
      k = 0; min = INFINITY;
      for (i = 0; i < n; i++)
          if (state[i].label == tentative && state[i].length < min) {
               min = state[i].length;
               k = i;
          }
      state[k].label = permanent;
  } while (k != s);

  /* Copy the path into the output array. */
  i = 0;  k = s;
  do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

Fig. 5-7. Dijkstra's algorithm to compute the shortest path through a graph.

5.2.4. Flow-Based Routing

The algorithms studied so far take only the topology into account. They do not consider the load. If, for example, there is always a huge amount of traffic from *A* to *B*, in Fig. 5-6, then it may be better to route traffic from *A* to *C* via *AGEFC*, even though this path is much longer than *ABC*. In this section we will study a static algorithm that uses both topology and load for routing. It is called **flow-based routing**.

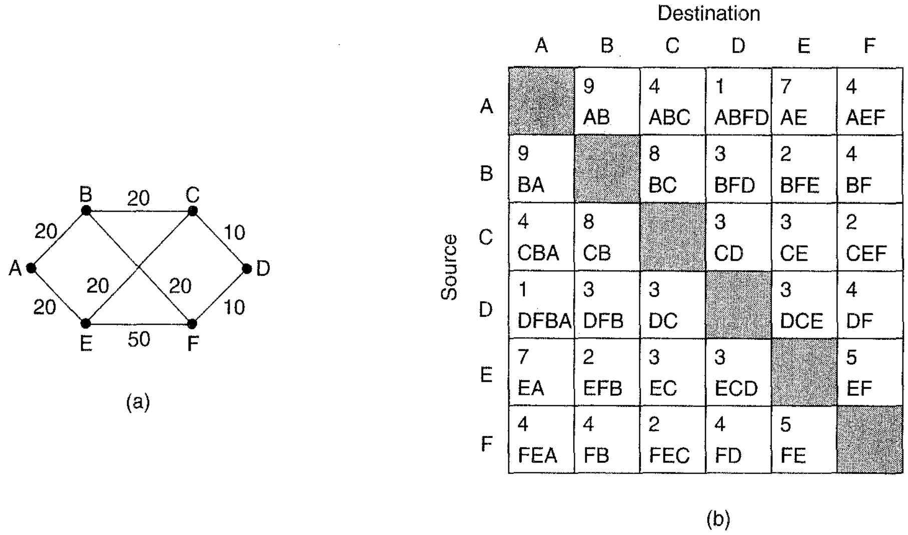
In some networks, the mean data flow between each pair of nodes is relatively stable and predictable. For example, in a corporate network for a retail store chain, each store might send orders, sales reports, inventory updates, and other well-defined types of messages to known sites in a predefined pattern, so that the total volume of traffic varies little from day to day. Under conditions in which the average traffic from *i* to *j* is known in advance and, to a reasonable approximation, constant in time, it is possible to analyze the flows mathematically to optimize the routing.

The basic idea behind the analysis is that for a given line, if the capacity and average flow are known, it is possible to compute the mean packet delay on that line from queueing theory. From the mean delays on all the lines, it is straightforward to calculate a flow-weighted average to get the mean packet delay for the whole subnet. The routing problem then reduces to finding the routing algorithm that produces the minimum average delay for the subnet. Fig. 5-8.

To use this technique, certain information must be known in advance. First the subnet topology must be known. Second, the traffic matrix, F_{ij}, must be given. Third, the line capacity matrix, C_{ij}, specifying the capacity of each line in bps must be available. Finally, a (possibly tentative) routing algorithm must be chosen.

As an example of this method, consider the full-duplex subnet of Fig. 5-8(a). The weights on the arcs give the capacities, C_{ij}, in each direction measured in kbps. The matrix of Fig. 5-8(b) has an entry for each source-destination pair. The entry for source *i* to destination *j* shows the route to be used for *i*-*j* traffic, and also the number of packets/sec to be sent from source *i* to destination *j*. For example, 3 packets/sec go from *B* to *D*, and they use route *BFD* to get there. Notice that some routing algorithm has already been applied to derive the routes shown in the matrix.

Given this information, it is straightforward to calculate the total in line *i*, λ_i. For example, the *B*-*D* traffic contributes 3 packets/sec to the *BF* line and also 3 packets/sec to the *FD* line. Similarly, the *A*-*D* traffic contributes 1 packet/sec to each of three lines. The total traffic in each eastbound line is shown in the λ_i column of Fig. 5-9. In this example, all the traffic is symmetric, that is, the *XY* traffic is identical to the *YX* traffic, for all *X* and *Y*. In real networks this condition does not always hold. The figure also shows the mean number of packets/sec on each line, μC_i assuming a mean packet size of $1/\mu = 800$ bits.



Source \ Destination	A	B	C	D	E	F
A	▨	9 AB	4 ABC	1 ABFD	7 AE	4 AEF
B	9 BA	▨	8 BC	3 BFD	2 BFE	4 BF
C	4 CBA	8 CB	▨	3 CD	3 CE	2 CEF
D	1 DFBA	3 DFB	3 DC	▨	3 DCE	4 DF
E	7 EA	2 EFB	3 EC	3 ECD	▨	5 EF
F	4 FEA	4 FB	2 FEC	4 FD	5 FE	▨

Fig. 5-8. (a) A subnet with line capacities shown in kbps. (b) The traffic in packets/sec and the routing matrix.

The next-to-last column of Fig. 5-9 gives the mean delay for each line derived from the queueing theory formula

$$T = \frac{1}{\mu C - \lambda}$$

where $1/\mu$ is the mean packet size in bits, C is the capacity in bps, and λ is the mean flow in packets/sec. For example, with a capacity $\mu C = 25$ packets/sec and an actual flow $\lambda = 14$ packets/sec, the mean delay is 91 msec. Note that with $\lambda = 0$, the mean delay is still 40 msec, because the capacity is 25 packets/sec. In other words, the "delay" includes both queueing and service time.

To compute the mean delay time for the entire subnet, we take the weighted sum of each of the eight lines, with the weight being the fraction of the total traffic using that line. In this example, the mean turns out to be 86 msec.

To evaluate a different routing algorithm, we can repeat the entire process, only with different flows to get a new average delay. If we restrict ourselves to only single path routing algorithms, as we have done so far, there are only a finite number of ways to route packets from each source to each destination. It is always possible to write a program to simply try them all, one after another, and find out which one has the smallest mean delay. Since this calculation can be done off-line in advance, the fact that it may be time consuming is not necessarily a serious problem. This one is then the best routing algorithm. Bertsekas and Gallager (1992) discuss flow-based routing in detail.

i	Line	λ_i (pkts/sec)	C_i (kbps)	μC_i (pkts/sec)	T_i (msec)	Weight
1	AB	14	20	25	91	0.171
2	BC	12	20	25	77	0.146
3	CD	6	10	12.5	154	0.073
4	AE	11	20	25	71	0.134
5	EF	13	50	62.5	20	0.159
6	FD	8	10	12.5	222	0.098
7	BF	10	20	25	67	0.122
8	EC	8	20	25	59	0.098

Fig. 5-9. Analysis of the subnet of Fig. 5-8 using a mean packet size of 800 bits. The reverse traffic (*BA*, *CB*, etc.) is the same as the forward traffic.

5.2.5. Distance Vector Routing

Modern computer networks generally use dynamic routing algorithms rather than the static ones described above. Two dynamic algorithms in particular, distance vector routing and link state routing, are the most popular. In this section we will look at the former algorithm. In the following one we will study the latter one.

Distance vector routing algorithms operate by having each router maintain a table (i.e, a vector) giving the best known distance to each destination and which line to use to get there. These tables are updated by exchanging information with the neighbors.

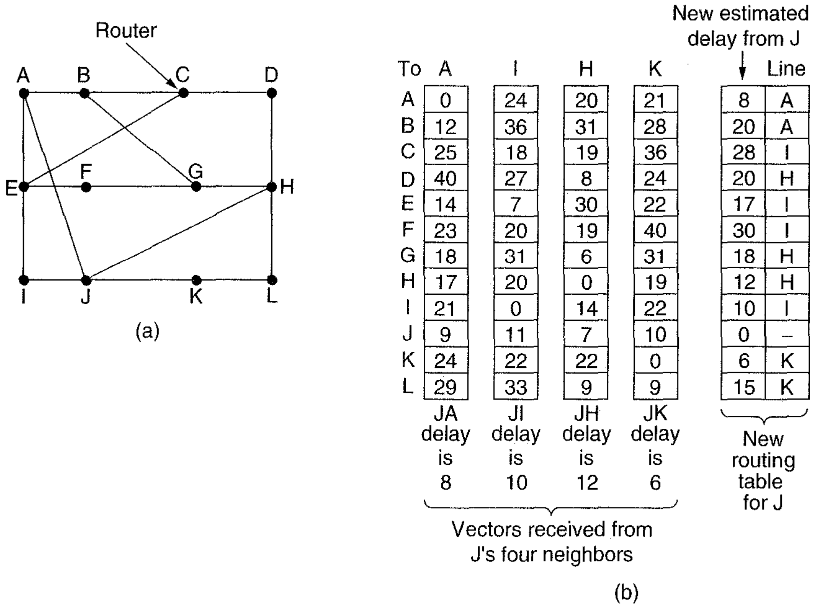
The distance vector routing algorithm is sometimes called by other names, including the distributed **Bellman-Ford** routing algorithm and the **Ford-Fulkerson** algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the name RIP and in early versions of DECnet and Novell's IPX. AppleTalk and Cisco routers use improved distance vector protocols.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in the subnet. This entry contains two parts: the preferred outgoing line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

The router is assumed to know the "distance" to each of its neighbors. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special ECHO packets that the receiver just timestamps and sends back as fast as it can.

As an example, assume that delay is used as a metric and that the router knows the delay to each of its neighbors. Once every T msec each router sends to each neighbor a list of its estimated delays to each destination. It also receives a similar list from each neighbor. Imagine that one of these tables has just come in from neighbor X, with X_i being X's estimate of how long it takes to get to router i. If the router knows that the delay to X is m msec, it also knows that it can reach router i via X in $X_i + m$ msec via X. By performing this calculation for each neighbor, a router can find out which estimate seems the best and use that estimate and the corresponding line in its new routing table. Note that the old routing table is not used in the calculation.

This updating process is illustrated in Fig. 5-10. Part (a) shows a subnet. The first four columns of part (b) show the delay vectors received from the neighbors of router *J*. *A* claims to have a 12-msec delay to *B*, a 25-msec delay to *C*, a 40-msec delay to *D*, etc. Suppose that *J* has measured or estimated its delay to its neighbors, *A*, *I*, *H*, and *K* as 8, 10, 12, and 6 msec, respectively.



To	A	I	H	K	New estimated delay from J	Line
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	–
K	24	22	22	0	6	K
L	29	33	9	9	15	K
	JA delay is 8	JI delay is 10	JH delay is 12	JK delay is 6		

Vectors received from J's four neighbors

New routing table for J

(b)

Fig. 5-10. (a) A subnet. (b) Input from *A*, *I*, *H*, *K*, and the new routing table for *J*.

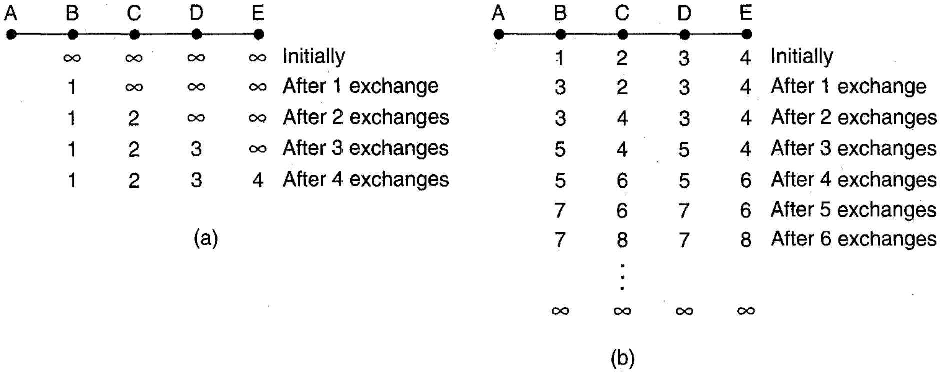
Consider how *J* computes its new route to router *G*. It knows that it can get to A in 8 msec, and *A* claims to be able to get to *G* in 18 msec, so *J* knows it can count on a delay of 26 msec to *G* if it forwards packets bound for *G* to *A*.

Similarly, it computes the delay to *G* via *I*, *H*, and *K* as 41 (31 + 10), 18 (6 + 12), and 37 (31 + 6) msec respectively. The best of these values is 18, so it makes an entry in its routing table that the delay to *G* is 18 msec, and that the route to use is via *H*. The same calculation is performed for all the other destinations, with the new routing table shown in the last column of the figure.

The Count-to-Infinity Problem

Distance vector routing works in theory but has a serious drawback in practice: although it converges to the correct answer, it may do so slowly. In particular, it reacts rapidly to good news, but leisurely to bad news. Consider a router whose best route to destination *X* is large. If on the next exchange neighbor *A* suddenly reports a short delay to *X*, the router just switches over to using the line to *A* to send traffic to *X*. In one vector exchange, the good news is processed.

To see how fast good news propagates, consider the five-node (linear) subnet of Fig. 5-11, where the delay metric is the number of hops. Suppose *A* is down initially and all the other routers know this. In other words, they have all recorded the delay to *A* as infinity.



A	B	C	D	E	
	∞	∞	∞	∞	Initially
	1	∞	∞	∞	After 1 exchange
	1	2	∞	∞	After 2 exchanges
	1	2	3	∞	After 3 exchanges
	1	2	3	4	After 4 exchanges

(a)

A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			
	∞	∞	∞	∞	

(b)

Fig. 5-11. The count-to-infinity problem.

When *A* comes up, the other routers learn about it via the vector exchanges. For simplicity we will assume that there is a gigantic gong somewhere that is struck periodically to initiate a vector exchange at all routers simultaneously. At the time of the first exchange, *B* learns that its left neighbor has zero delay to *A*. *B* now makes an entry in its routing table that *A* is one hop away to the left. All the other routers still think that *A* is down. At this point, the routing table entries for *A* are as shown in the second row of Fig. 5-11(a). On the next exchange, *C* learns that *B* has a path of length 1 to *A*, so it updates its routing table to indicate a path of length 2, but *D* and *E* do not hear the good news until later. Clearly, the good news is spreading at the rate of one hop per exchange. In a subnet whose longest path is of length *N* hops, within *N* exchanges everyone will know about newly revived lines and routers.

Now let us consider the situation of Fig. 5-11(b), in which all the lines and routers are initially up. Routers *B*, *C*, *D*, and *E* have distances to *A* of 1, 2, 3, and 4, respectively. Suddenly *A* goes down, or alternatively, the line between *A* and *B* is cut, which is effectively the same thing from *B*'s point of view.

At the first packet exchange, *B* does not hear anything from *A*. Fortunately, *C* says "Do not worry. I have a path to *A* of length 2." Little does *B* know that *C*'s path runs through *B* itself. For all *B* knows, *C* might have ten outgoing lines all with independent paths to *A* of length 2. As a result, *B* now thinks it can reach *A* via *C*, with a path length of 3. *D* and *E* do not update their entries for *A* on the first exchange.

On the second exchange, *C* notices that each of its neighbors claims to have a path to *A* of length 3. It picks one of the them at random and makes its new distance to *A* 4, as shown in the third row of Fig. 5-11(b). Subsequent exchanges produce the history shown in the rest of Fig. 5-11(b).

From this figure, its should be clear why bad news travels slowly: no router ever has a value more than one higher than the minimum of all its neighbors. Gradually, all the routers work their way up to infinity, but the number of exchanges required depends on the numerical value used for infinity. For this reason, it is wise to set infinity to the longest path plus 1. If the metric is time delay, there is no well-defined upper bound, so a high value is needed to prevent a path with a long delay from being treated as down. Not entirely surprisingly, this problem is known as the **count-to-infinity** problem.

The Split Horizon Hack

Many ad hoc solutions to the count-to-infinity problem have been proposed in the literature, each one more complicated and less useful than the one before it. We will describe just one of them here and then tell why it, too, fails. The **split horizon** algorithm works the same way as distance vector routing, except that the distance to *X* is not reported on the line that packets for *X* are sent on (actually, it is reported as infinity). In the initial state of Fig. 5-11(b), for example, *C* tells *D* the truth about the distance to *A*, but *C* tells *B* that its distance to *A* is infinite. Similarly, *D* tells the truth to *E* but lies to *C*.

Now let us see what happens when *A* goes down. On the first exchange, *B* discovers that the direct line is gone, and *C* is reporting an infinite distance to *A* as well. Since neither of its neighbors can get to *A*, *B* sets its distance to infinity as well. On the next exchange, *C* hears that *A* is unreachable from both of its neighbors, so it marks *A* as unreachable too. Using split horizon, the bad news propagates one hop per exchange. This rate is much better than without split horizon.

The real bad news is that split horizon, although widely used, sometimes fails.

Consider, for example, the four-node subnet of Fig. 5-12. Initially, both *A* and *B* have a distance 2 to *D*, and *C* has a distance 1 there.

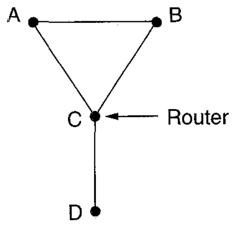


Fig. 5-12. An example where split horizon fails.

Now suppose that the *CD* line goes down. Using split horizon, both *A* and *B* tell *C* that they cannot get to *D*. Thus *C* immediately concludes that *D* is unreachable and reports this to both *A* and *B*. Unfortunately, *A* hears that *B* has a path of length 2 to *D*, so it assumes it can get to *D* via *B* in 3 hops. Similarly, *B* concludes it can get to *D* via *A* in 3 hops. On the next exchange, they each set their distance to *D* to 4. Both of them gradually count to infinity, precisely the behavior we were trying to avoid.

5.2.6. Link State Routing

Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing. Two primary problems caused its demise. First, since the delay metric was queue length, it did not take line bandwidth into account when choosing routes. Initially, all the lines were 56 kbps, so line bandwidth was not an issue, but after some lines had been upgraded to 230 kbps and others to 1.544 Mbps, not taking bandwidth into account was a major problem. Of course, it would have been possible to change the delay metric to factor in line bandwidth, but a second problem also existed, namely, the algorithm often took too long to converge, even with tricks like split horizon. For these reasons, it was replaced by an entirely new algorithm now called **link state routing**. Variants of link state routing are now widely used.

The idea behind link state routing is simple and can be stated as five parts. Each router must

1. Discover its neighbors and learn their network addresses.
2. Measure the delay or cost to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to all other routers.
5. Compute the shortest path to every other router.

In effect, the complete topology and all delays are experimentally measured and distributed to every router. Then Dijkstra's algorithm can be used to find the shortest path to every other router. Below we will consider each of these five steps in more detail.

Learning about the Neighbors

When a router is booted, its first task is to learn who its neighbors are. It accomplishes this goal by sending a special HELLO packet on each point-to-point line. The router on the other end is expected to send back a reply telling who it is. These names must be globally unique because when a distant router later hears that three routers are all connected to *F*, it is essential that it can determine whether or not all three mean the same *F*.

When two or more routers are connected by a LAN, the situation is slightly more complicated. Fig. 5-13(a) illustrates a LAN to which three routers, *A*, *C*, and *F*, are directly connected. Each of these routers is connected to one or more additional routers, as shown.

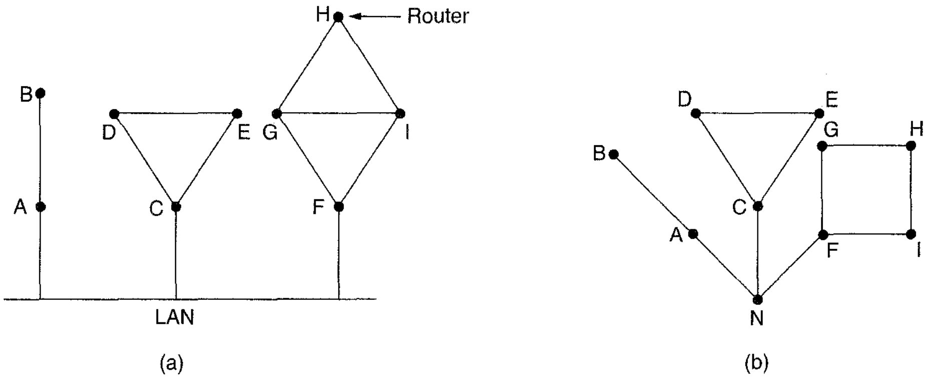


Fig. 5-13. (a) Nine routers and a LAN. (b) A graph model of (a).

One way to model the LAN is to consider it as a node itself, as shown in Fig. 5-13(b). Here we have introduced a new, artificial node, *N*, to which *A*, *C*, and *F* are connected. The fact that it is possible to go from *A* to *C* on the LAN is represented by the path *ANC* here.

Measuring Line Cost

The link state routing algorithm requires each router to know, or at least have a reasonable estimate, of the delay to each of its neighbors. The most direct way to determine this delay is to send a special ECHO packet over the line that the other side is required to send back immediately. By measuring the round-trip time and dividing it by two, the sending router can get a reasonable estimate of the delay. For even better results, the test can be conducted several times, and the average used.

An interesting issue is whether or not to take the load into account when measuring the delay. To factor the load in, the round-trip timer must be started when the ECHO packet is queued. To ignore the load, the timer should be started when the ECHO packet reaches the front of the queue.

Arguments can be made both ways. Including traffic-induced delays in the measurements means that when a router has a choice between two lines with the same bandwidth, one of which is heavily loaded all the time and one of which is not, it will regard the route over the unloaded line as a shorter path. This choice will result in better performance.

Unfortunately, there is also an argument against including the load in the delay calculation. Consider the subnet of Fig. 5-14, which is divided up into two parts, East and West, connected by two lines, *CF* and *EI*. Suppose that most of the traffic between East and West is using line *CF*, and as a result, this line is heavily loaded with long delays. Including queueing delay in the shortest path calculation will make *EI* more attractive. After the new routing tables have been installed, most of the East-West traffic will now go over *EI*, overloading this line. Consequently, in the next update, *CF* will appear to be the shortest path. As a result, the routing tables may oscillate wildly, leading to erratic routing and many potential problems. If load is ignored and only bandwidth is considered, this problem does not occur. Alternatively, the load can be spread over both lines, but this solution does not fully utilize the best path.

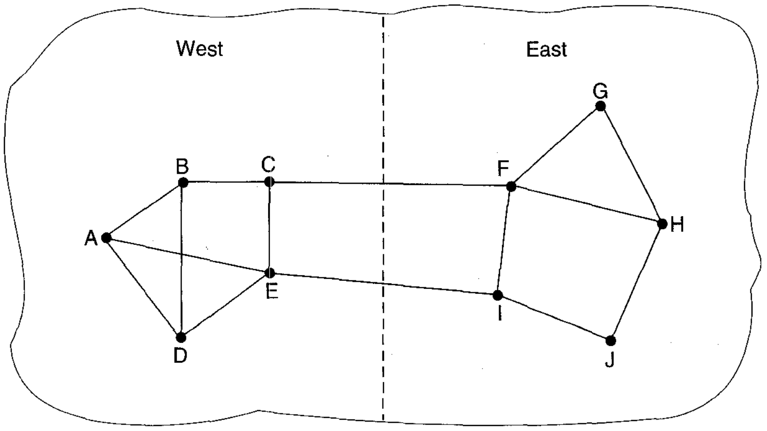
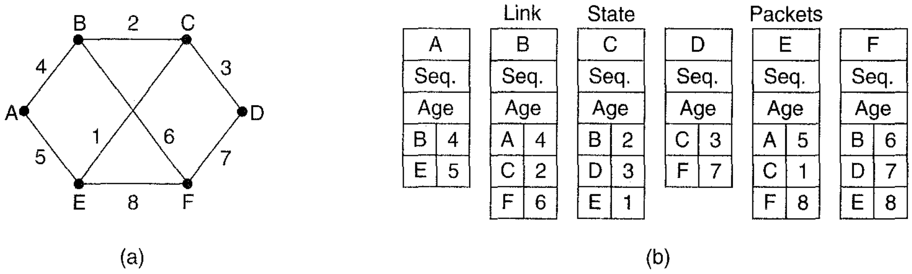


Fig. 5-14. A subnet in which the East and West parts are connected by two lines.

Building Link State Packets

Once the information needed for the exchange has been collected, the next step is for each router to build a packet containing all the data. The packet starts with the identity of the sender, followed by a sequence number and age (to be described later), and a list of neighbors. For each neighbor, the delay to that neighbor is given. An example subnet is given in Fig. 5-15(a) with delays shown in the lines. The corresponding link state packets for all six routers are shown in Fig. 5-15(b).



Link State Packets

A	B	C	D	E	F
Seq.	Seq.	Seq.	Seq.	Seq.	Seq.
Age	Age	Age	Age	Age	Age
B 4	A 4	B 2	C 3	A 5	B 6
E 5	C 2	D 3	F 7	C 1	D 7
	F 6	E 1		F 8	E 8

Fig. 5-15. (a) A subnet. (b) The link state packets for this subnet.

Building the link state packets is easy. The hard part is determining when to build them. One possibility is to build them periodically, that is, at regular intervals. Another possibility is when some significant event occurs, such as a line or neighbor going down or coming back up again, or changing its properties appreciably.

Distributing the Link State Packets

The trickiest part of the algorithm is distributing the link state packets reliably. As the packets are distributed and installed, the routers getting the first ones will change their routes. Consequently, the different routers may be using different versions of the topology, which can lead to inconsistencies, loops, unreachable machines, and other problems.

First we will describe the basic distribution algorithm. Later we will give some refinements. The fundamental idea is to use flooding to distribute the link state packets. To keep the flood in check, each packet contains a sequence number that is incremented for each new packet sent. Routers keep track of all the (source router, sequence) pairs they see. When a new link state packet comes in, it is checked against the list of packets already seen. If it is new, it is forwarded on all lines except the one it arrived on. If it is a duplicate, it is discarded.

If a packet with a sequence number lower than the highest one seen so far ever arrives, it is rejected as being obsolete.

This algorithm has a few problems, but they are manageable. First, if the sequence numbers wrap around, confusion will reign. The solution here is to use a 32-bit sequence number. With one link state packet per second, it would take 137 years to wrap around, so this possibility can be ignored.

Second, if a router ever crashes, it will lose track of its sequence number. If it starts again at 0, the next packet will be rejected as a duplicate.

Third, if a sequence number is ever corrupted and 65,540 is received instead of 4 (a 1-bit error), packets 5 through 65,540 will be rejected as obsolete, since the current sequence number is thought to be 65,540.

The solution to all these problems is to include the age of each packet after the sequence number and decrement it once per second. When the age hits zero, the information from that router is discarded. Normally, a new packet comes in, say, every 10 minutes, so router information only times out when a router is down (or six consecutive packets have been lost, an unlikely event). The age field is also decremented by each router during the initial flooding process, to make sure no packet can get lost and live for an indefinite period of time (a packet whose age is zero is discarded).

Some refinements to this algorithm make it more robust. When a link state packet comes in to a router for flooding, it is not queued for transmission immediately. Instead it is put in a holding area to wait a short while first. If another link state packet from the same source comes in before it is transmitted, their sequence numbers are compared. If they are equal, the duplicate is discarded. If they are different, the older one is thrown out. To guard against errors on the router-router lines, all link state packets are acknowledged. When a line goes idle, the holding area is scanned in round robin order to select a packet or acknowledgement to send.

The data structure used by router *B* for the subnet shown in Fig. 5-15(a) is depicted in Fig. 5-16. Each row here corresponds to a recently arrived, but as yet not fully processed, link state packet. The table records where the packet originated, its sequence number and age, and the data. In addition, there are send and acknowledgement flags for each of *B*'s three lines (to *A*, *C*, and *F*, respectively). The send flags mean that the packet must be sent on the indicated line. The acknowledgement flags mean that it must be acknowledged there.

In Fig. 5-16, the link state packet from *A* arrived directly, so it must be sent to *C* and *F* and acknowledged to *A*, as indicated by the flag bits. Similarly, the packet from *F* has to be forwarded to *A* and *C* and acknowledged to *F*.

However, the situation with the third packet, from *E*, is different. It arrived twice, once via *EAB* and once via *EFB*. Consequently, it has to be sent only to *C*, but acknowledged to both *A* and *F*, as indicated by the bits.

If a duplicate arrives while the original is still in the buffer, bits have to be changed. For example, if a copy of *C*'s state arrives from *F* before the fourth

			Send flags			ACK flags			
Source	Seq.	Age	A	C	F	A	C	F	Data
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

Fig. 5-16. The packet buffer for router *B* in Fig. 5-15.

entry in the table has been forwarded, the six bits will be changed to 100011 to indicate that the packet must be acknowledged to *F* but not sent there.

Computing the New Routes

Once a router has accumulated a full set of link state packets, it can construct the entire subnet graph because every link is represented. Every link is, in fact, represented twice, once for each direction. The two values can be averaged or used separately.

Now Dijkstra's algorithm can be run locally to construct the shortest path to all possible destinations. The results of this algorithm can be installed in the routing tables, and normal operation resumed.

For a subnet with *n* routers, each of which has *k* neighbors, the memory required to store the input data is proportional to *kn*. For large subnets, this can be a problem. Also, the computation time can be an issue. Nevertheless, in many practical situations, link state routing works well.

However, problems with the hardware or software can wreak havoc with this algorithm (also with other ones). For example, if a router claims to have a line it does not have, or forgets a line it does have, the subnet graph will be incorrect. If a router fails to forward packets, or corrupts them while forwarding them, trouble will arise. Finally, if it runs out of memory or does the routing calculation wrong, bad things will happen. As the subnet grows into the range of tens or hundreds of thousands of nodes, the probability of some router failing occasionally becomes nonnegligible. The trick is to try to arrange to limit the damage when the inevitable happens. Perlman (1988) discusses these problems and their solutions in detail.

Link state routing is widely used in actual networks, so a few words about some example protocols using it are in order. The OSPF protocol, which is increasingly being used in the Internet, uses a link state algorithm. We will describe OSPF in Sec. 5.5.5.

Another important link state protocol is **IS-IS (Intermediate System-Intermediate System)**, which was designed for DECnet and later adopted by ISO for use with its connectionless network layer protocol, CLNP. Since then it has been modified to handle other protocols as well, most notably, IP. IS-IS is used in numerous Internet backbones (including the old NSFNET backbone), and in some digital cellular systems such as CDPD. Novell NetWare uses a minor variant of IS-IS (NLSP) for routing IPX packets.

Basically IS-IS distributes a picture of the router topology, from which the shortest paths are computed. Each router announces, in its link state information, which network layer addresses it can reach directly. These addresses can be IP, IPX, AppleTalk, or any other addresses. IS-IS can even support multiple network layer protocols at the same time.

Many of the innovations designed for IS-IS were adopted by OSPF (OSPF was designed several years after IS-IS). These include a self-stabilizing method of flooding link state updates, the concept of a designated router on a LAN, and the method of computing and supporting path splitting and multiple metrics. As a consequence, there is very little difference between IS-IS and OSPF. The most important difference is that IS-IS is encoded in such a way that it is easy and natural to simultaneously carry information about multiple network layer protocols, a feature OSPF does not have. This advantage is especially valuable in large multiprotocol environments.

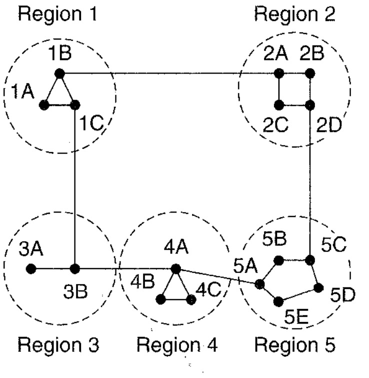
5.2.7. Hierarchical Routing

As networks grow in size, the router routing tables grow proportionally. Not only is router memory consumed by ever increasing tables, but more CPU time is needed to scan them and more bandwidth is needed to send status reports about them. At a certain point the network may grow to the point where it is no longer feasible for every router to have an entry for every other router, so the routing will have to be done hierarchically, as it is in the telephone network.

When hierarchical routing is used, the routers are divided into what we will call **regions**, with each router knowing all the details about how to route packets to destinations within its own region, but knowing nothing about the internal structure of other regions. When different networks are connected together, it is natural to regard each one as a separate region in order to free the routers in one network from having to know the topological structure of the other ones.

For huge networks, a two-level hierarchy may be insufficient; it may be necessary to group the regions into clusters, the clusters into zones, the zones into groups, and so on, until we run out of names for aggregations. As an example of a multilevel hierarchy, consider how a packet might be routed from Berkeley, California to Malindi, Kenya. The Berkeley router would know the detailed topology within California but would send all out-of-state traffic to the Los Angeles router. The Los Angeles router would be able to route traffic to other domestic routers, but would send foreign traffic to New York. The New York router would be programmed to direct all traffic to the router in the destination country responsible for handling foreign traffic, say in Nairobi. Finally, the packet would work its way down the tree in Kenya until it got to Malindi.

Figure 5-17 gives a quantitative example of routing in a two-level hierarchy with five regions. The full routing table for router 1*A* has 17 entries, as shown in Fig. 5-17(b). When routing is done hierarchically, as in Fig. 5-17(c), there are entries for all the local routers as before, but all other regions have been condensed into a single router, so all traffic for region 2 goes via the 1*B*−2*A* line, but the rest of the remote traffic goes via the 1*C*−3*B* line. Hierarchical routing has reduced the table from 17 to 7 entries. As the ratio of the number of regions to the number of routers per region grows, the savings in table space increase.



(a)

Full table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

Hierarchical table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

(c)

Fig. 5-17. Hierarchical routing.

Unfortunately, these gains in space are not free. There is a penalty to be paid, and this penalty is in the form of increased path length. For example, the best route from 1*A* to 5*C* is via region 2, but with hierarchical routing all traffic to region 5 goes via region 3, because that is better for most destinations in region 5.

When a single network becomes very large, an interesting question is: How many levels should the hierarchy have? For example, consider a subnet with 720 routers. If there is no hierarchy, each router needs 720 routing table entries. If the subnet is partitioned into 24 regions of 30 routers each, each router needs 30 local entries plus 23 remote entries for a total of 53 entries. If a three-level hierarchy is chosen, with eight clusters, each containing 9 regions of 10 routers, each router needs 10 entries for local routers, 8 entries for routing to other regions within its own cluster, and 7 entries for distant clusters, for a total of 25 entries. Kamoun and Kleinrock (1979) have discovered that the optimal number of levels for an N router subnet is $\ln N$, requiring a total of $e \ln N$ entries per router. They have also shown that the increase in effective mean path length caused by hierarchical routing is sufficiently small that it is usually acceptable.

5.2.8. Routing for Mobile Hosts

Millions of people have portable computers nowadays, and they generally want to read their email and access their normal file systems wherever in the world they may be. These mobile hosts introduce a new complication: to route a packet to a mobile host, the network first has to find it. The subject of incorporating mobile hosts into a network is very young, but in this section we will sketch some of the issues here and give a possible solution.

The model of the world that network designers typically use is shown in Fig. 5-18. Here we have a WAN consisting of routers and hosts. Connected to the WAN are LANs, MANs, and wireless cells of the type we studied in Chap. 2.

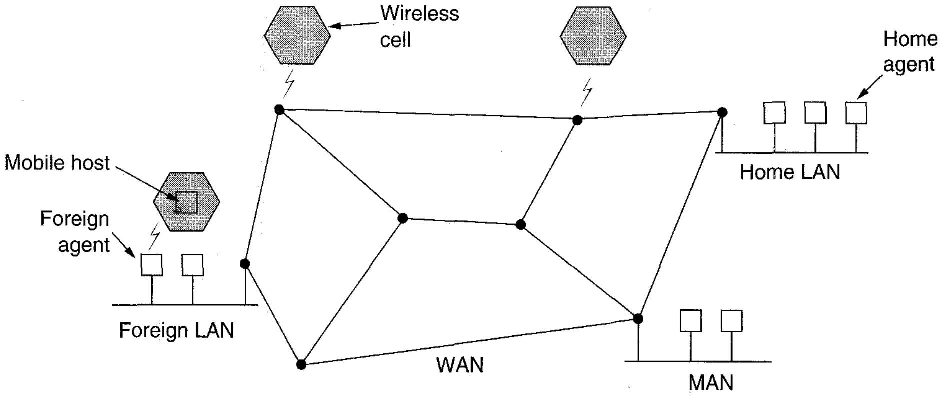


Fig. 5-18. A WAN to which LANs, MANs, and wireless cells are attached.

Users who never move are said to be stationary. They are connected to the network by copper wires or fiber optics. In contrast, we can distinguish two other kinds of users. Migratory users are basically stationary users who move from one fixed site to another from time to time but use the network only when they are physically connected to it. Roaming users actually compute on the run and want to maintain their connections as they move around. We will use the term **mobile users** to mean either of the latter two categories, that is, all users who are away from home.

All users are assumed to have a permanent **home location** that never changes. Users also have a permanent home address that can be used to determine their home locations, analogous to the way the telephone number 1-212-5551212 indicates the United States (country code 1) and Manhattan (212). The routing goal in systems with mobile users is to make it possible to send packets to mobile users using their home addresses, and have the packets efficiently reach them wherever they may be. The trick, of course, is to find them.

In the model of Fig. 5-18, the world is divided up (geographically) into small units. Let us call them areas, where an area is typically a LAN or wireless cell. Each area has one or more **foreign agents**, which keep track of all mobile users visiting the area. In addition, each area has a **home agent**, which keeps track of users whose home is in the area, but who are currently visiting another area.

When a new user enters an area, either by connecting to it (e.g., plugging into the LAN), or just wandering into the cell, his computer must register itself with the foreign agent there. The registration procedure typically works like this:

1. Periodically, each foreign agent broadcasts a packet announcing its existence and address. A newly arrived mobile host may wait for one of these messages, but if none arrives quickly enough, the mobile host can broadcast a packet saying: "Are there any foreign agents around?"

2. The mobile host registers with the foreign agent, giving its home address, current data link layer address, and some security information.

3. The foreign agent contacts the mobile host's home agent and says: "One of your hosts is over here." The message from the foreign agent to the home agent contains the foreign agent's network address. It also includes the security information, to convince the home agent that the mobile host is really there.

4. The home agent examines the security information, which contains a timestamp, to prove that it was generated within the past few seconds. If it is happy, it tells the foreign agent to proceed.

5. When the foreign agent gets the acknowledgement from the home agent, it makes an entry in its tables and informs the mobile host that it is now registered.

Ideally, when a user leaves an area, that, too, should be announced to allow deregistration, but many users abruptly turn off their computers when done.

When a packet is sent to a mobile user, it is routed to the user's home LAN because that is what the address says should be done, as illustrated in step 1 of Fig. 5-19. Packets sent to the mobile user on its home LAN are intercepted by the home agent. The home agent then looks up the mobile user's new (temporary) location and finds the address of the foreign agent handling the mobile user. The home agent then does two things. First, it encapsulates the packet in the payload field of an outer packet and sends the latter to the foreign agent (step 2 in Fig. 5-19). This mechanism is called tunneling; we will look at it in more detail later. After getting the encapsulated packet, the foreign agent removes the original packet from the payload field and sends it to the mobile user as a data link frame.

Second, the home agent tells the sender to henceforth send packets to the mobile host by encapsulating them in the payload of packets explicitly addressed to the foreign agent, instead of just sending them to the mobile user's home address (step 3). Subsequent packets can now be routed directly to the user via the foreign agent (step 4), bypassing the home location entirely.

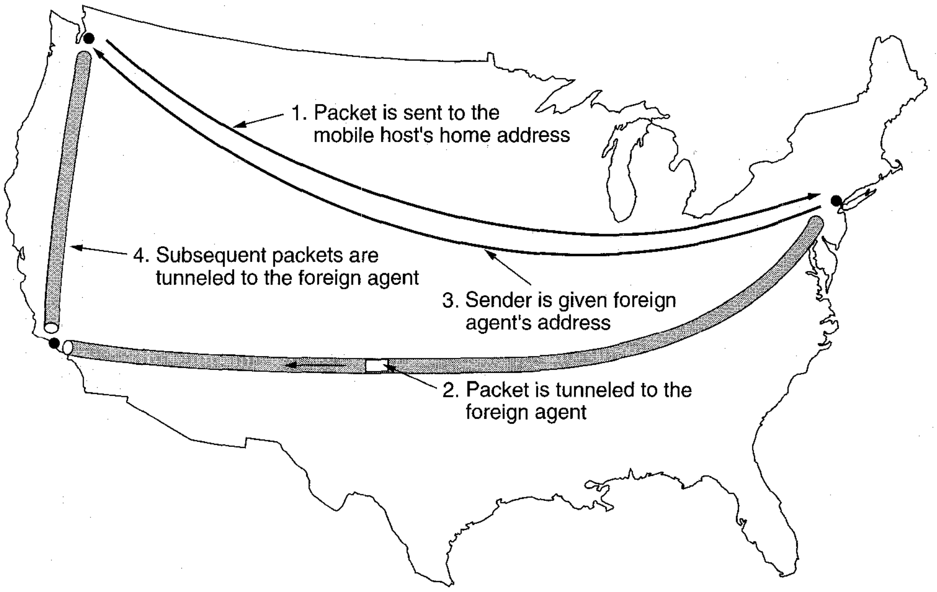

1. Packet is sent to the mobile host's home address
4. Subsequent packets are tunneled to the foreign agent
3. Sender is given foreign agent's address
2. Packet is tunneled to the foreign agent

Fig. 5-19. Packet routing for mobile users.

The various schemes that have been proposed differ in several ways. First, there is the issue of how much of this protocol is carried out by the routers and how much by the hosts, and in the latter case, by which layer in the hosts. Second, a few schemes, routers along the way record mapped addresses so they can intercept and redirect traffic even before it gets to the home location. Third, in some schemes each visitor is given a unique temporary address; in others, the temporary address refers to an agent that handles traffic for all visitors.

Fourth, the schemes differ in how they actually manage to arrange for packets that are addressed to one destination to be delivered to a different one. One choice is changing the destination address and just retransmitting the modified packet. Alternatively, the whole packet, home address and all, can be encapsulated inside the payload of another packet sent to the temporary address. Finally, the schemes differ in their security aspects. In general, when a host or router gets a message of the form "Starting right now, please send all of Cayla's mail to me," it might have a couple of questions about whom it was talking to and whether or not this is a good idea. Several mobile host protocols are discussed and compared in (Ioannidis and Maguire, 1993; Myles and Skellern, 1993; Perkins, 1993; Teraoka et al., 1993; and Wada et al., 1993).

5.2.9. Broadcast Routing

For some applications, hosts need to send messages to many or all other hosts. For example, a service distributing weather reports, stock market updates, or live radio programs might work best by broadcasting to all machines and letting those that are interested read the data. Sending a packet to all destinations simultaneously is called **broadcasting**; various methods have been proposed for doing it.

One broadcasting method that requires no special features from the subnet is for the source to simply send a distinct packet to each destination. Not only is the method wasteful of bandwidth, but it also requires the source to have a complete list of all destinations. In practice this may be the only possibility, but it is the least desirable of the methods.

Flooding is another obvious candidate. Although flooding is ill-suited for ordinary point-to-point communication, for broadcasting it might rate serious consideration, especially if none of the methods described below are applicable. The problem with flooding as a broadcast technique is the same problem it has as a point-to-point routing algorithm: it generates too many packets and consumes too much bandwidth.

A third algorithm is **multidestination routing**. If this method is used, each packet contains either a list of destinations or a bit map indicating the desired destinations. When a packet arrives at a router, the router checks all the destinations to determine the set of output lines that will be needed. (An output line is needed if it is the best route to at least one of the destinations.) The router generates a new copy of the packet for each output line to be used and includes in each packet only those destinations that are to use the line. In effect, the destination set is partitioned among the output lines. After a sufficient number of hops, each packet will carry only one destination and can be treated as a normal packet. Multidestination routing is like separately addressed packets, except that when several packets must follow the same route, one of them pays full fare and the rest ride free.

A fourth broadcast algorithm makes explicit use of the sink tree for the router initiating the broadcast, or any other convenient spanning tree for that matter. A

spanning tree is a subset of the subnet that includes all the routers but contains no loops. If each router knows which of its lines belong to the spanning tree, it can copy an incoming broadcast packet onto all the spanning tree lines except the one it arrived on. This method makes excellent use of bandwidth, generating the absolute minimum number of packets necessary to do the job. The only problem is that each router must have knowledge of some spanning tree for it to be applicable. Sometimes this information is available (e.g., with link state routing) but sometimes it is not (e.g., with distance vector routing).

Our last broadcast algorithm is an attempt to approximate the behavior of the previous one, even when the routers do not know anything at all about spanning trees. The idea is remarkably simple once it has been pointed out. When a broadcast packet arrives at a router, the router checks to see if the packet arrived on the line that is normally used for sending packets *to* the source of the broadcast. If so, there is an excellent chance that the broadcast packet itself followed the best route from the router and is therefore the first copy to arrive at the router. This being the case, the router forwards copies of it onto all lines except the one it arrived on. If, however, the broadcast packet arrived on a line other than the preferred one for reaching the source, the packet is discarded as a likely duplicate.

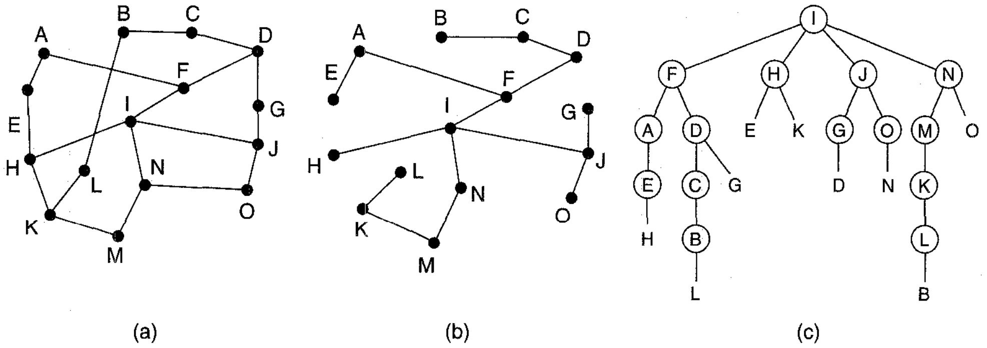


Fig. 5-20. Reverse path forwarding. (a) A subnet. (b) A spanning tree. (c) The tree built by reverse path forwarding.

An example of the algorithm, called **reverse path forwarding**, is shown in Fig. 5-20. Part (a) shows a subnet, part (b) shows a sink tree for router *I* of that subnet, and part (c) shows how the reverse path algorithm works. On the first hop, *I* sends packets to *F*, *H*, *J*, and *N*, as indicated by the second row of the tree. Each of these packets arrives on the preferred path to *I* (assuming that the preferred path falls along the sink tree) and is so indicated by a circle around the letter. On the second hop, eight packets are generated, two by each of the routers that received a packet on the first hop. As it turns out, all eight of these arrive at previously unvisited routers, and five of these arrive along the preferred line. Of the six packets generated on the third hop, only three arrive on the preferred path (at *C*, *E*, and *K*); the others are duplicates. After five hops and 23 packets, the broadcasting terminates, compared with four hops and 14 packets had the sink tree been followed exactly.

The principal advantage of reverse path forwarding is that it is both reasonably efficient and easy to implement. It does not require routers to know about spanning trees, nor does it have the overhead of a destination list or bit map in each broadcast packet as does multidestination addressing. Nor does it require any special mechanism to stop the process, as flooding does (either a hop counter in each packet and a priori knowledge of the subnet diameter, or a list of packets already seen per source).

5.2.10. Multicast Routing

For some applications, widely-separated processes work together in groups, for example, a group of processes implementing a distributed database system. It frequently is necessary for one process to send a message to all the other members of the group. If the group is small, it can just send each other member a point-to-point message. If the group is large, this strategy is expensive. Sometimes broadcasting can be used, but using broadcasting to inform 1000 machines on a million-node network is inefficient because most receivers are not interested in the message (or worse yet, they are definitely interested but are not supposed to see it). Thus we need a way to send messages to well-defined groups that are numerically large in size but small compared to the network as a whole.

Sending a message to such a group is called **multicasting**, and its routing algorithm is called **multicast routing**. In this section we will describe one way of doing multicast routing. For additional information, see (Deering and Cheriton, 1990; Deering et al., 1994; and Rajagopalan, 1992).

To do multicasting, group management is required. Some way is needed to create and destroy groups, and for processes to join and leave groups. How these tasks are accomplished is not of concern to the routing algorithm. What is of concern is that when a process joins a group, it informs its host of this fact. It is important that routers know which of their hosts belong to which groups. Either hosts must inform their routers about changes in group membership, or routers must query their hosts periodically. Either way, routers learn about which of their hosts are in which groups. Routers tell their neighbors, so the information propagates through the subnet.

To do multicast routing, each router computes a spanning tree covering all other routers in the subnet. For example, in Fig. 5-21(a) we have a subnet with two groups, 1 and 2. Some routers are attached to hosts that belong to one or both of these groups, as indicated in the figure. A spanning tree for the leftmost router is shown in Fig. 5-21(b).

When a process sends a multicast packet to a group, the first router examines its spanning tree and prunes it, removing all lines that do not lead to hosts that are

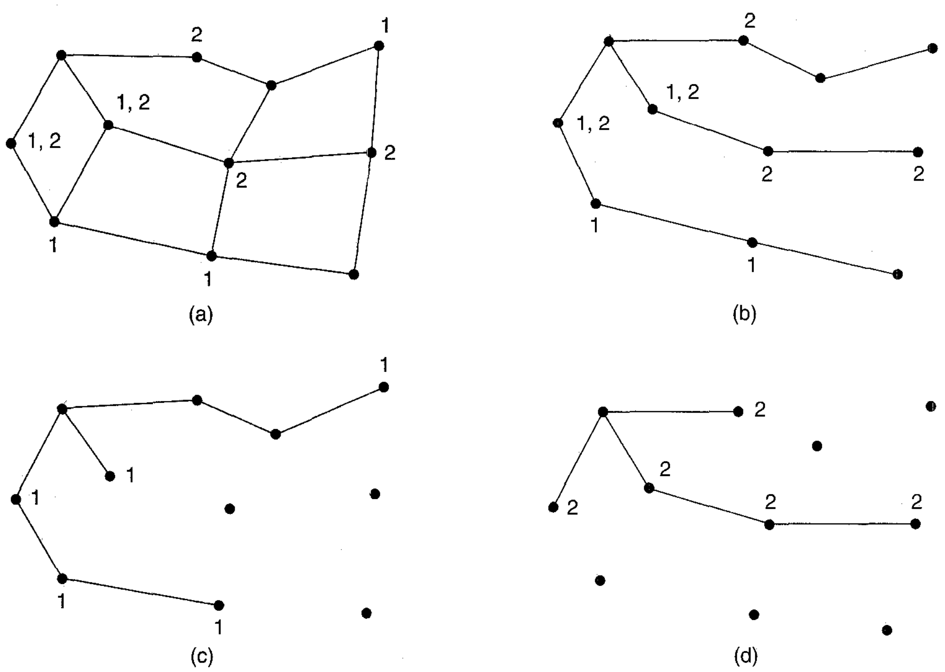


Fig. 5-21. (a) A subnet. (b) A spanning tree for the leftmost router. (c) A multicast tree for group 1. (d) A multicast tree for group 2.

members of the group. In our example, Fig. 5-21(c) shows the pruned spanning tree for group 1. Similarly, Fig. 5-21(d) shows the pruned spanning tree for group 2. Multicast packets are forwarded only along the appropriate spanning tree.

Various ways of pruning the spanning tree are possible. The simplest one can be used if link state routing is used, and each router is aware of the complete subnet topology, including which hosts belong to which groups. Then the spanning tree can be pruned by starting at the end of each path and working toward the root, removing all routers that do not belong to the group in question.

With distance vector routing, a different pruning strategy can be followed. The basic algorithm is reverse path forwarding. However, whenever a router with no hosts interested in a particular group and no connections to other routers receives a multicast message for that group, it responds with a PRUNE message, telling the sender not to send it any more multicasts for that group. When a router with no group members among its own hosts has received such messages on all its lines, it, too, can respond with a PRUNE message. In this way, the subnet is recursively pruned.

One potential disadvantage of this algorithm is that it scales poorly to large networks. Suppose that a network has *n* groups, each with an average of *m* members. For each group, *m* pruned spanning trees must be stored, for a total of *mn* trees. When many large groups exist, considerable storage is needed to store all the trees.

An alternative design uses **core-base trees** (Ballardie et al., 1993). Here, a single spanning tree per group is computed, with the root (the core) near the middle of the group. To send a multicast message, a host sends it to the core, which then does the multicast along the spanning tree. Although this tree will not be optimal for all sources, the reduction in storage costs from *m* trees to one tree per group is a major saving.

5.3. CONGESTION CONTROL ALGORITHMS

When too many packets are present in (a part of) the subnet, performance degrades. This situation is called **congestion**. Figure 5-22 depicts the symptom. When the number of packets dumped into the subnet by the hosts is within its carrying capacity, they are all delivered (except for a few that are afflicted with transmission errors), and the number delivered is proportional to the number sent. However, as traffic increases too far, the routers are no longer able to cope, and they begin losing packets. This tends to make matters worse. At very high traffic, performance collapses completely, and almost no packets are delivered.

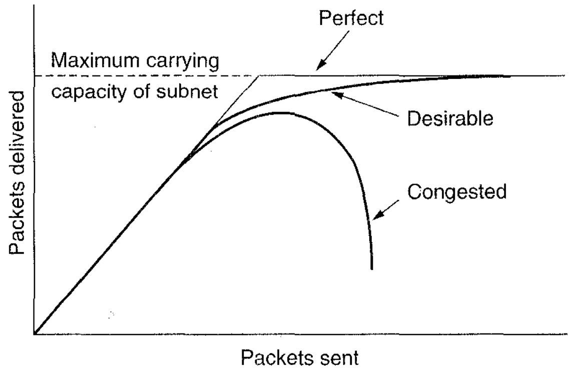


Fig. 5-22. When too much traffic is offered, congestion sets in and performance degrades sharply.

Congestion can be brought about by several factors. If all of a sudden, streams of packets begin arriving on three or four input lines and all need the same output line, a queue will build up. If there is insufficient memory to hold all of them, packets will be lost. Adding more memory may help up to a point, but Nagle (1987) discovered that if routers have an infinite amount of memory, congestion gets worse, not better, because by the time packets get to the front of the queue, they have already timed out (repeatedly), and duplicates have been sent. All these packets will be dutifully forwarded to the next router, increasing the load all the way to the destination.

Slow processors can also cause congestion. If the routers' CPUs are slow at performing the bookkeeping tasks required of them (queueing buffers, updating tables, etc.), queues can build up, even though there is excess line capacity. Similarly, low-bandwidth lines can also cause congestion. Upgrading the lines but not changing the processors, or vice versa, often helps a little, but frequently just shifts the bottleneck. Also, upgrading part, but not all, of the system, often just moves the bottleneck somewhere else. The real problem is frequently a mismatch between parts of the system. This problem will persist until all the components are in balance.

Congestion tends to feed upon itself and become worse. If a router has no free buffers, it must ignore newly arriving packets. When a packet is discarded, the sending router (a neighbor) may time out and retransmit it, perhaps ultimately many times. Since it cannot discard the packet until it has been acknowledged, congestion at the receiver's end forces the sender to refrain from releasing a buffer it would have normally freed. In this manner, congestion backs up, like cars approaching a toll booth.

It is worth explicitly pointing out the difference between congestion control and flow control, as the relationship is subtle. Congestion control has to do with making sure the subnet is able to carry the offered traffic. It is a global issue, involving the behavior of all the hosts, all the routers, the store-and-forwarding processing within the routers, and all the other factors that tend to diminish the carrying capacity of the subnet.

Flow control, in contrast, relates to the point-to-point traffic between a given sender and a given receiver. Its job is to make sure that a fast sender cannot continually transmit data faster than the receiver can absorb it. Flow control nearly always involves some direct feedback from the receiver to the sender to tell the sender how things are doing at the other end.

To see the difference between these two concepts, consider a fiber optic network with a capacity of 1000 gigabits/sec on which a supercomputer is trying to transfer a file to a personal computer at 1 Gbps. Although there is no congestion (the network itself is not in trouble), flow control is needed to force the supercomputer to stop frequently to give the personal computer a chance to breathe.

At the other extreme, consider a store-and-forward network with 1-Mbps lines and 1000 large computers, half of which are trying to transfer files at 100 kbps to the other half. Here the problem is not that of fast senders overpowering slow receivers, but simply that the total offered traffic exceeds what the network can handle.

The reason congestion control and flow control are often confused is that some congestion control algorithms operate by sending messages back to the various sources telling them to slow down when the network gets into trouble. Thus a host can get a "slow down" message either because the receiver cannot handle the load, or because the network cannot handle it. We will come back to this point later.

We will start our study of congestion control by looking at a general model for dealing with it. Then we will look at broad approaches to preventing it in the first place. After that, we will look at various dynamic algorithms for coping with it once it has set in.

5.3.1. General Principles of Congestion Control

Many problems in complex systems, such as computer networks, can be viewed from a control theory point of view. This approach leads to dividing all solutions into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made.

Tools for doing open-loop control include deciding when to accept new traffic, deciding when to discard packets and which ones, and making scheduling decisions at various points in the network. All of these have in common the fact that they make decisions without regard to the current state of the network.

In contrast, closed loop solutions are based on the concept of a feedback loop. This approach has three parts when applied to congestion control:

1. Monitor the system to detect when and where congestion occurs.
2. Pass this information to places where action can be taken.
3. Adjust system operation to correct the problem.

Various metrics can be used to monitor the subnet for congestion. Chief among these are the percentage of all packets discarded for lack of buffer space, the average queue lengths, the number of packets that time out and are retransmitted, the average packet delay, and the standard deviation of packet delay. In all cases, rising numbers indicate growing congestion.

The second step in the feedback loop is to transfer the information about the congestion from the point where it is detected to the point where something can be done about it. The obvious way is for the router detecting the congestion to send a packet to the traffic source or sources, announcing the problem. Of course, these extra packets increase the load at precisely the moment that more load is not needed, namely, when the subnet is congested.

However, other possibilities also exist. For example, a bit or field can be reserved in every packet for routers to fill in whenever congestion gets above some threshold level. When a router detects this congested state, it fills in the field in all outgoing packets, to warn the neighbors.

Still another approach is to have hosts or routers send probe packets out periodically to explicitly ask about congestion. This information can then be used to route traffic around problem areas. Some radio stations have helicopters flying around their cities to report on road congestion in the hope that their listeners will route their packets (cars) around hot spots.

In all feedback schemes, the hope is that knowledge of congestion will cause the hosts to take appropriate action to reduce the congestion. To work correctly, the time scale must be adjusted carefully. If every time two packets arrive in a row, a router yells STOP, and every time a router is idle for 20 μsec it yells GO, the system will oscillate wildly and never converge. On the other hand, if it waits 30 minutes to make sure before saying anything, the congestion control mechanism will react too sluggishly to be of any real use. To work well, some kind of averaging is needed, but getting the time constant right is a nontrivial matter.

Many congestion control algorithms are known. To provide a way to organize them in a sensible way, Yang and Reddy (1995) have developed a taxonomy for congestion control algorithms. They begin by dividing all algorithms into open loop or closed loop, as described above. They further divide the open loop algorithms into ones that act at the source versus ones that act at the destination. The closed loop algorithms are also divided into two subcategories: explicit feedback versus implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithms, the source deduces the existence of congestion by making local observations, such as the time needed for acknowledgements to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. Two solutions come to mind: increase the resources or decrease the load. For example, the subnet may start using dial-up telephone lines to temporarily increase the bandwidth between certain points. In systems like SMDS (see Chap. 1), it may ask the carrier for additional bandwidth for a while. On satellite systems, increasing transmission power often gives higher bandwidth. Splitting traffic over multiple routes instead of always using the best one may also effectively increase the bandwidth. Finally, spare routers that are normally used only as backups (to make the system fault tolerant) can be put on-line to give more capacity when serious congestion appears.

However, sometimes it is not possible to increase the capacity, or it has already been increased to the limit. The only way then to beat back the congestion is to decrease the load. Several ways exist to reduce the load, including denying service to some users, degrading service to some or all users, and having users schedule their demands in a more predictable way.

Some of these methods, which we will study shortly, can best be applied to virtual circuits. For subnets that use virtual circuits internally, these methods can be used at the network layer. For datagram subnets, they can nevertheless sometimes be used on transport layer connections. In this chapter, we will focus on their use in the network layer. In the next one, we will see what can be done at the transport layer to manage congestion.

5.3.2. Congestion Prevention Policies

Let us begin our study of methods to control congestion by looking at open loop systems. These systems are designed to minimize congestion in the first place, rather than letting it happen and reacting after the fact. They try to achieve their goal by using appropriate policies at various levels. In Fig. 5-23 we see different data link, network, and transport policies that can affect congestion (Jain, 1990).

Layer	Policies
Transport	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy • Timeout determination
Network	• Virtual circuits versus datagram inside the subnet • Packet queueing and service policy • Packet discard policy • Routing algorithm • Packet lifetime management
Data link	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy

Fig. 5-23. Policies that affect congestion.

Let us start at the data link layer and work our way upward. The retransmission policy deals with how fast a sender times out and what it transmits upon timeout. A jumpy sender that times out quickly and retransmits all outstanding packets using go back n will put a heavier load on the system than a leisurely sender that uses selective repeat. Closely related to this is caching policy. If receivers routinely discard all out-of-order packets, these packets will have to be transmitted again later, creating extra load.

Acknowledgement policy also affects congestion. If each packet is acknowledged immediately, the acknowledgement packets generate extra traffic. However, if acknowledgements are saved up to piggyback onto reverse traffic, extra timeouts and retransmissions may result. A tight flow control scheme (e.g., a small window) reduces the data rate and thus helps fight congestion.

At the network layer, the choice between virtual circuits and datagrams affects congestion, since many congestion control algorithms work only with virtual circuit subnets. Packet queueing and service policy relates to whether routers have one queue per input line, one queue per output line, or both. It also relates to the order packets are processed (e.g., round robin, or priority based). Discard policy is the rule telling which packet is dropped when there is no space. A good policy can help alleviate congestion and a bad one can make it worse.

The routing algorithm can help avoid congestion by spreading the traffic over all the lines, whereas a bad one can send too much traffic over already congested lines. Finally, packet lifetime management deals with how long a packet may live before being discarded. If it is too long, lost packets may clog up the works for a long time, but if it is too short, packets may sometimes time out before reaching their destination, thus inducing retransmissions.

In the transport layer, the same issues occur as in the data link layer, but in addition, determining the timeout interval is harder because the transit time across the network is less predictable than the transit time over a wire between two routers. If it is too short, extra packets will be sent unnecessarily. If it is too long, congestion will be reduced, but the response time will suffer whenever a packet is lost.

5.3.3. Traffic Shaping

One of the main causes of congestion is that traffic is often bursty. If hosts could be made to transmit at a uniform rate, congestion would be less common. Another open loop method to help manage congestion is forcing the packets to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called **traffic shaping**.

Traffic shaping is about regulating the average *rate* (and burstiness) of data transmission. In contrast, the sliding window protocols we studied earlier limit the amount of data in transit at once, not the rate at which it is sent. When a virtual circuit is set up, the user and the subnet (i.e., the customer and the carrier) agree on a certain traffic pattern (i.e., shape) for that circuit. As long as the customer fulfills her part of the bargain and only sends packets according to the agreed upon contract, the carrier promises to deliver them all in a timely fashion. Traffic shaping reduces congestion and thus helps the carrier live up to its promise. Such agreements are not so important for file transfers but are of great importance for real-time data, such as audio and video connections, which do not tolerate congestion well.

In effect, with traffic shaping the customer says to the carrier: "My transmission pattern will look like this. Can you handle it?" If the carrier agrees, the issue arises of how the carrier can tell if the customer is following the agreement, and what to do if the customer is not. Monitoring a traffic flow is called **traffic policing**. Agreeing to a traffic shape and policing it afterward are easier with virtual circuit subnets than with datagram subnets. However, even with datagram subnets, the same ideas can be applied to transport layer connections.

The Leaky Bucket Algorithm

Imagine a bucket with a small hole in the bottom, as illustrated in Fig. 5-24(a). No matter at what rate water enters the bucket, the outflow is at a constant rate, ρ, when there is any water in the bucket, and zero when the bucket is empty. Also, once the bucket is full, any additional water entering it spills over the sides and is lost (i.e., does not appear in the output stream under the hole).

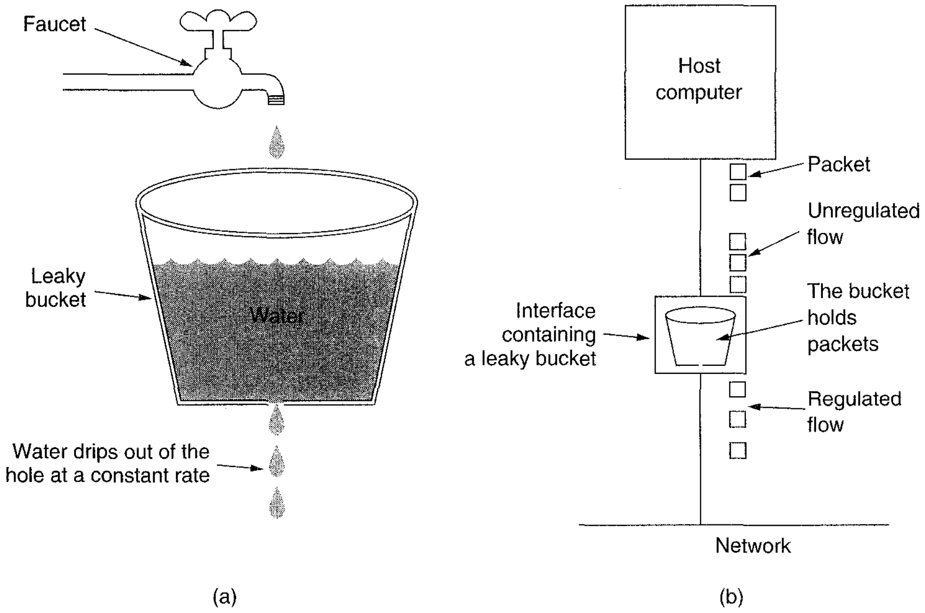


Fig. 5-24. (a) A leaky bucket with water. (b) A leaky bucket with packets.

The same idea can be applied to packets, as shown in Fig. 5-24(b). Conceptually, each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more processes within the host try to send a packet when the maximum number are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulated by the host operating system. It was first proposed by Turner (1986) and is called the **leaky bucket algorithm**. In fact, it is nothing other than a single-server queueing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. Again, this can be enforced by the interface card or by the operating system. This mechanism turns an uneven flow of packets from the user processes inside the host into an even flow of packets onto the network, smoothing out bursts and greatly reducing the chances of congestion.

When the packets are all the same size (e.g., ATM cells), this algorithm can be used as described. However, when variable-sized packets are being used, it is often better to allow a fixed number of bytes per tick, rather than just one packet. Thus if the rule is 1024 bytes per tick, a single 1024-byte packet can be admitted on a tick, two 512-byte packets, four 256-byte packets, and so on. If the residual byte count is too low, the next packet must wait until the next tick.

Implementing the original leaky bucket algorithm is easy. The leaky bucket consists of a finite queue. When a packet arrives, if there is room on the queue it is appended to the queue; otherwise, it is discarded. At every clock tick, one packet is transmitted (unless the queue is empty).

The byte-counting leaky bucket is implemented almost the same way. At each tick, a counter is initialized to n. If the first packet on the queue has fewer bytes than the current value of the counter, it is transmitted, and the counter is decremented by that number of bytes. Additional packets may also be sent, as long as the counter is high enough. When the counter drops below the length of the next packet on the queue, transmission stops until the next tick, at which time the residual byte count is overwritten and lost.

As an example of a leaky bucket, imagine that a computer can produce data at 25 million bytes/sec (200 Mbps) and that the network also runs at this speed. However, the routers can handle this data rate only for short intervals. For long intervals, they work best at rates not exceeding 2 million bytes/sec. Now suppose data comes in 1-million-byte bursts, one 40-msec burst every second. To reduce the average rate to 2 MB/sec, we could use a leaky bucket with $\rho = 2$ MB/sec and a capacity, C, of 1 MB. This means that bursts of up to 1 MB can be handled without data loss, and that such bursts are spread out over 500 msec, no matter how fast they come in.

In Fig. 5-25(a) we see the input to the leaky bucket running at 25 MB/sec for 40 msec. In Fig. 5-25(b) we see the output draining out at a uniform rate of 2 MB/sec for 500 msec.

The Token Bucket Algorithm

The leaky bucket algorithm enforces a rigid output pattern at the average rate, no matter how bursty the traffic is. For many applications, it is better to allow the output to speed up somewhat when large bursts arrive, so a more flexible algorithm is needed, preferably one that never loses data. One such algorithm is the **token bucket algorithm**. In this algorithm, the leaky bucket holds tokens, generated by a clock at the rate of one token every ΔT sec. In Fig. 5-26(a) we see a bucket holding three tokens, with five packets waiting to be transmitted. For a packet to be transmitted, it must capture and destroy one token. In Fig. 5-26(b)

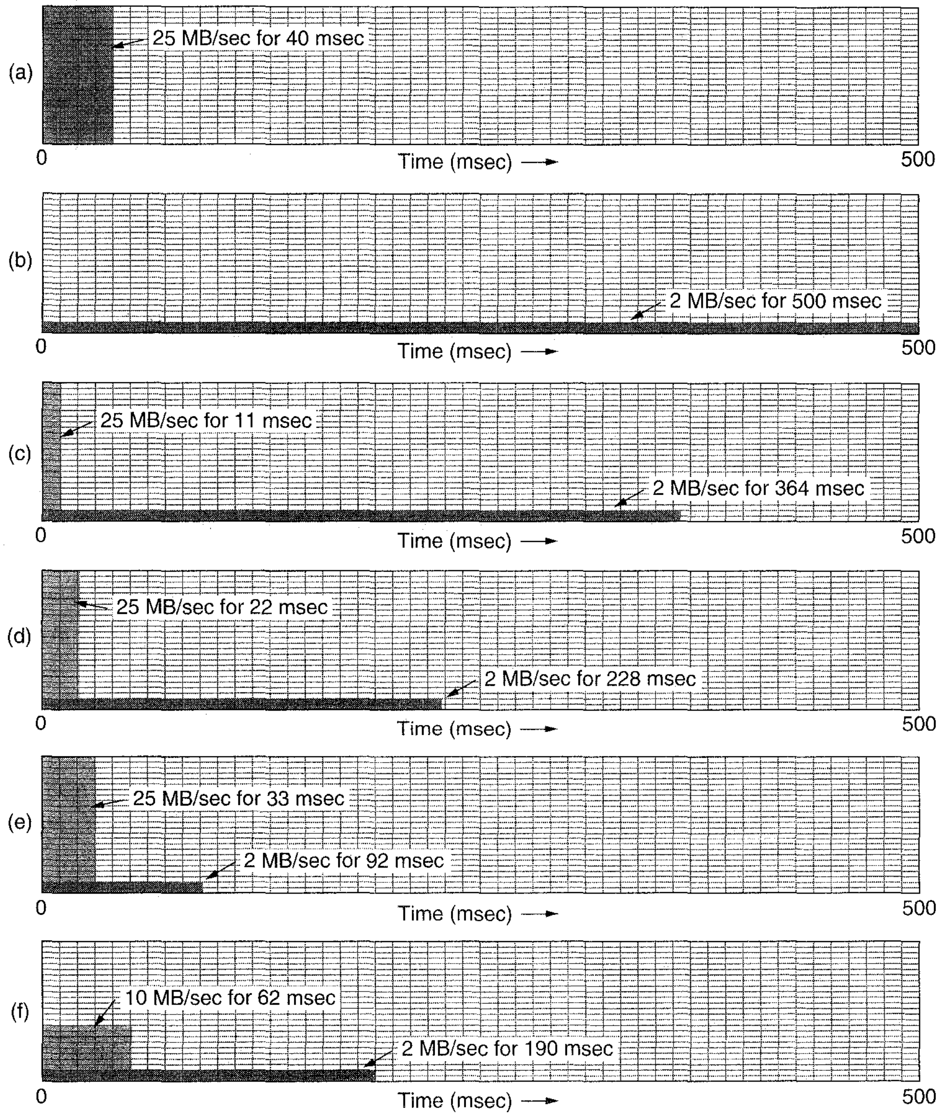


Fig. 5-25. (a) Input to a leaky bucket. (b) Output from a leaky bucket. (c) - (e) Output from a token bucket with capacities of 250KB, 500KB, and 750KB. (f) Output from a 500KB token bucket feeding a 10 MB/sec leaky bucket.

we see that three of the five packets have gotten through, but the other two are stuck waiting for two more tokens to be generated.

The token bucket algorithm provides a different kind of traffic shaping than the leaky bucket algorithm. The leaky bucket algorithm does not allow idle hosts

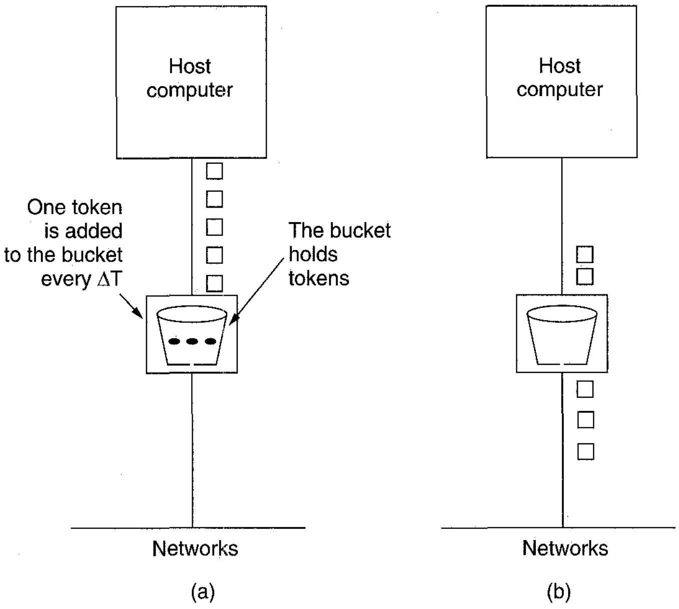


Fig. 5-26. The token bucket algorithm. (a) Before. (b) After.

to save up permission to send large bursts later. The token bucket algorithm does allow saving, up to the maximum size of the bucket, n. This property means that bursts of up to n packets can be sent at once, allowing some burstiness in the output stream and giving faster response to sudden bursts of input.

Another difference between the two algorithms is that the token bucket algorithm throws away tokens when the bucket fills up but never discards packets. In contrast, the leaky bucket algorithm discards packets when the bucket fills up.

Here too, a minor variant is possible, in which each token represents the right to send not one packet, but k bytes. A packet can only be transmitted if enough tokens are available to cover its length in bytes. Fractional tokens are kept for future use.

The leaky bucket and token bucket algorithms can also be used to smooth traffic between routers, as well as being used to regulate host output as in our examples. However, one clear difference is that a token bucket regulating a host can make the host stop sending when the rules say it must. Telling a router to stop sending while its input keeps pouring in may result in lost data.

The implementation of the basic token bucket algorithm is just a variable that counts tokens. The counter is incremented by one every ΔT and decremented by one whenever a packet is sent. When the counter hits zero, no packets may be sent. In the byte-count variant, the counter is increment by k bytes every ΔT and decremented by the length of each packet sent.

Essentially what the token bucket does is allow bursts, but up to a regulated maximum length. Look at Fig. 5-25(c) for example. Here we have a token bucket with a capacity of 250 KB. Tokens arrive at a rate allowing output at 2 MB/sec. Assuming the token bucket is full when the 1-MB burst arrives, the bucket can drain at the full 25 MB/sec for about 11 msec. Then it has to cut back to 2 MB/sec until the entire input burst has been sent.

Calculating the length of the maximum rate burst is slightly tricky. It is not just 1 MB divided by 25 MB/sec because while the burst is being output, more tokens arrive. If we call the burst length S sec, the token bucket capacity C bytes, the token arrival rate ρ bytes/sec, and the maximum output rate M bytes/sec, we see that an output burst contains a maximum of $C + \rho S$ bytes. We also know that the number of bytes in a maximum-speed burst of length S seconds is MS. Hence we have

$$C + \rho S = MS$$

We can solve this equation to get $S = C/(M - \rho)$. For our parameters of $C = 250$ KB, $M = 25$ MB/sec, and $\rho = 2$ MB/sec, we get a burst time of about 11 msec. Figure 5-25(d) and Fig. 5-25(e) show the token bucket for capacities of 500-KB and 750 KB, respectively.

A potential problem with the token bucket algorithm is that it allows large bursts again, even though the maximum burst interval can be regulated by careful selection of ρ and M. Frequently it is desirable to reduce the peak rate, but without going back to the low value of the original leaky bucket.

One way to get smoother traffic is to put a leaky bucket after the token bucket. The rate of the leaky bucket should be higher than the token bucket's ρ but lower than the maximum rate of the network. Figure 5-25(f) shows the output for a 500 KB token bucket followed by a 10-MB/sec leaky bucket.

Policing all these schemes can be a bit tricky. Essentially, the network has to simulate the algorithm and make sure that no more packets or bytes are being sent than are permitted. Excess packets are then discarded or downgraded, as discussed later.

5.3.4. Flow Specifications

Traffic shaping is most effective when the sender, receiver, and subnet all agree to it. To get agreement, it is necessary to specify the traffic pattern in a precise way. Such an agreement is called a **flow specification**. It consists of a data structure that describes both the pattern of the injected traffic and the quality of service desired by the applications. A flow specification can apply either to the packets sent on a virtual circuit, or to a sequence of datagrams sent between a source and a destination (or even to multiple destinations).

In this section we will describe an example flow specification designed by Partridge (1992). It is shown in Fig. 5-27. The idea is that before a connection is established or before a sequence of datagrams are sent, the source gives the flow specification to the subnet for approval. The subnet can either accept it, reject it, or come back with a counterproposal ("I cannot give you 100 msec average delay; can you live with 150 msec?"). Once the sender and subnet have struck a deal, the sender can ask the receiver if it, too, agrees.

Characteristics of the Input	Service Desired
Maximum packet size (bytes)	Loss sensitivity (bytes)
Token bucket rate (bytes/sec)	Loss interval (µsec)
Token bucket size (bytes)	Burst loss sensitivity (packets)
Maximum transmission rate (bytes/sec)	Minimum delay noticed (µsec)
	Maximum delay variation (µsec)
	Quality of guarantee

Fig. 5-27. An example flow specification.

Let us now examine the parameters of our example flow specification starting with the traffic specification. The *Maximum packet size* tells how big packets may be. The next two parameters implicitly assume that traffic will be shaped by the token bucket algorithm working in bytes. They tell how many bytes are put into the token bucket per second, and how big the bucket is. If the rate is r bytes/sec and the bucket size is b bytes, then in any arbitrary time interval Δt, the maximum number of bytes that may be sent is $b + r\Delta t$. Here the first term represents the maximum possible contents of the bucket at the start of the interval and the second one represents the new credits that come in during the interval. The *Maximum transmission rate* is the top rate the host is capable of producing under any conditions and implicitly specifies the shortest time interval in which the token bucket could be emptied.

The second column specifies what the application wants from the subnet. The first and second parameters represent the numerator and denominator of a fraction giving the maximum acceptable loss rate (e.g., 1 byte per hour). Alternatively, they can indicate that the flow is insensitive to packet loss. The *Burst loss sensitivity* tells how many consecutive lost packets can be tolerated.

The next two service parameters deal with delay. The *Minimum delay noticed* says how long a packet can be delayed without the application noticing. For a file transfer, it might be a second, but for an audio stream 3 msec might be the limit. The *Maximum delay variation* tries to quantify the fact that some applications are not sensitive to the actual delay but are highly sensitive to the **jitter**, that is, the amount of variation in the end-to-end packet transit time. It is two times the number of microseconds a packet's delay may vary from the average. Thus a value of 2000 means that a packet may be up to 1 msec early or late, but no more.

Finally, the *Quality of guarantee* indicates whether or not the application really means it. On the one hand, the loss and delay characteristics might be ideal goals, but no harm is done if they are not met. On the other hand, they might be so important that if they cannot be met, the application simply terminates. Intermediate positions are also possible.

Although we have looked at the flow specification as a request from the application to the subnet, it can also be a return value telling what the subnet can do. Thus it can potentially be used for an extended negotiation about the service level.

A problem inherent with any flow specification is that the application may not know what it really wants. For example, an application program running in New York might be quite happy with a delay of 200 msec to Sydney, but most unhappy with the same 200-msec delay to Boston. Here the "minimum service" is clearly a function of what is thought to be possible.

5.3.5. Congestion Control in Virtual Circuit Subnets

The congestion control methods described above are basically open loop: they try to prevent congestion from occurring in the first place, rather than dealing with it after the fact. In this section we will describe some approaches to dynamically controlling congestion in virtual circuit subnets. In the next two, we will look at techniques that can be used in any subnet.

One technique that is widely used to keep congestion that has already started from getting worse is **admission control**. The idea is simple: once congestion has been signaled, no more virtual circuits are set up until the problem has gone away. Thus, attempts to set up new transport layer connections fail. Letting more people in just makes matters worse. While this approach is crude, it is simple and easy to carry out. In the telephone system, when a switch gets overloaded, it also practices admission control, by not giving dial tones.

An alternative approach is to allow new virtual circuits but carefully route all new virtual circuits around problem areas. For example, consider the subnet of Fig. 5-28(a), in which two routers are congested, as indicated.

Suppose that a host attached to router *A* wants to set up a connection to a host attached to router *B*. Normally, this connection would pass through one of the congested routers. To avoid this situation, we can redraw the subnet as shown in Fig. 5-28(b), omitting the congested routers and all of their lines. The dashed line shows a possible route for the virtual circuit that avoids the congested routers.

Another strategy relating to virtual circuits is to negotiate an agreement between the host and subnet when a virtual circuit is set up. This agreement normally specifies the volume and shape of the traffic, quality of service required, and other parameters. To keep its part of the agreement, the subnet will typically reserve resources along the path when the circuit is set up. These resources can include table and buffer space in the routers and bandwidth on the lines. In this

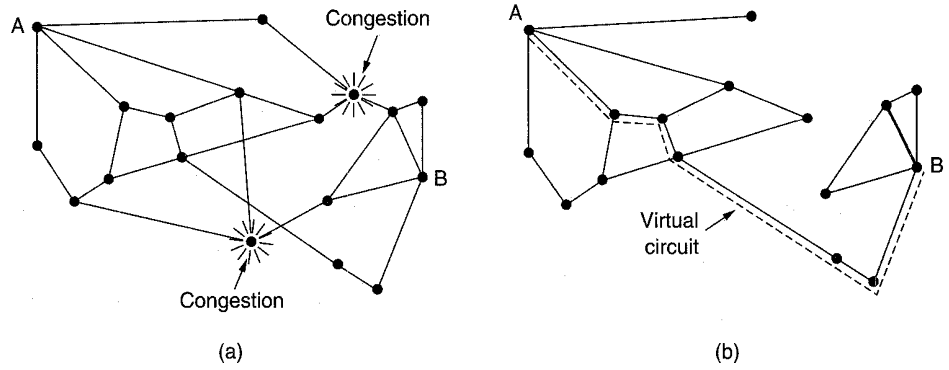


Fig. 5-28. (a) A congested subnet. (b) A redrawn subnet that eliminates the congestion and a virtual circuit from *A* to *B*.

way, congestion is unlikely to occur on the new virtual circuits because all the necessary resources are guaranteed to be available.

This kind of reservation can be done all the time as standard operating procedure, or only when the subnet is congested. A disadvantage of doing it all the time is that it tends to waste resources. If six virtual circuits that might use 1 Mbps all pass through the same physical 6-Mbps line, the line has to be marked as full, even though it may rarely happen that all six virtual circuits are transmitting at the same time. Consequently, the price of the congestion control is unused bandwidth.

5.3.6. Choke Packets

Let us now turn to an approach that can be used in both virtual circuit and datagram subnets. Each router can easily monitor the utilization of its output lines and other resources. For example, it can associate with each line a real variable, u, whose value, between 0.0 and 1.0, reflects the recent utilization of that line. To maintain a good estimate of u, a sample of the instantaneous line utilization, f (either 0 or 1), can be made periodically and u updated according to

$$u_{\text{new}} = au_{\text{old}} + (1 - a)f$$

where the constant a determines how fast the router forgets recent history.

Whenever u moves above the threshold, the output line enters a "warning" state. Each newly arriving packet is checked to see if its output line is in warning state. If so, the router sends a **choke packet** back to the source host, giving it the destination found in the packet. The original packet is tagged (a header bit is turned on) so that it will not generate any more choke packets further along the path and is then forwarded in the usual way.

When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination by X percent. Since other packets aimed at the same destination are probably already under way and will generate yet more choke packets, the host should ignore choke packets referring to that destination for a fixed time interval. After that period has expired, the host listens for more choke packets for another interval. If one arrives, the line is still congested, so the host reduces the flow still more and begins ignoring choke packets again. If no choke packets arrive during the listening period, the host may increase the flow again. The feedback implicit in this protocol can help prevent congestion yet not throttle any flow unless trouble occurs.

Hosts can reduce traffic by adjusting their policy parameters, for example, window size or leaky bucket output rate. Typically, the first choke packet causes the data rate to be reduced to 0.50 of its previous rate, the next one causes a reduction to 0.25, and so on. Increases are done in smaller increments to prevent congestion from reoccurring quickly.

Several variations on this congestion control algorithm have been proposed. For one, the routers can maintain several thresholds. Depending on which threshold has been crossed, the choke packet can contain a mild warning, a stern warning, or an ultimatum.

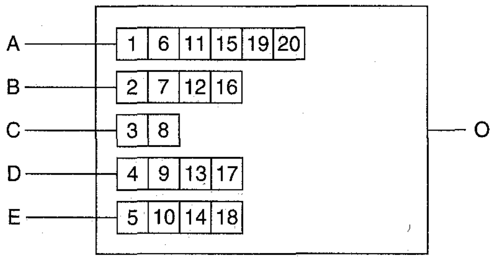
Another variation is to use queue lengths or buffer utilization instead of line utilization as the trigger signal. The same exponential weighting can be used with this metric as with u, of course.

Weighted Fair Queueing

A problem with using choke packets is that the action to be taken by the source hosts is voluntary. Suppose that a router is being swamped by packets from four sources, and it sends choke packets to all of them. One of them cuts back, as it is supposed to, but the other three just keep blasting away. The result is that the honest host gets an even smaller share of the bandwidth than it had before.

To get around this problem, and thus make compliance more attractive, Nagle (1987) proposed a **fair queueing** algorithm. The essence of the algorithm is that routers have multiple queues for each output line, one for each source. When a line becomes idle, the router scans the queues round robin, taking the first packet on the next queue. In this way, with n hosts competing for a given output line, each host gets to send one out of every n packets. Sending more packets will not improve this fraction. Some ATM switches use this algorithm.

Although a start, the algorithm has a problem: it gives more bandwidth to hosts that use large packets than to hosts that use small packets. Demers et al. (1990) suggested an improvement in which the round robin is done in such a way as to simulate a byte-by-byte round robin, instead of a packet-by-packet round robin. In effect, it scans the queues repeatedly, byte-for-byte, until it finds the tick on which each packet will be finished. The packets are then sorted in order of their finishing and sent in that order. The algorithm is illustrated in Fig. 5-29.



(a)

Packet	Finishing time
C	8
B	16
D	17
E	18
A	20

(b)

Fig. 5-29. (a) A router with five packets queued for line *O*. (b) Finishing times for the five packets.

In Fig. 5-29(a) we see packets of length 2 to 6 bytes. At (virtual) clock tick 1, the first byte of the packet on line *A* is sent. Then goes the first byte of the packet on line *B*, and so on. The first packet to finish is *C*, after eight ticks. The sorted order is given in Fig. 5-29(b). In the absence of new arrivals, the packets will be sent in the order listed, from *C* to *A*.

One problem with this algorithm is that it gives all hosts the same priority. In many situations, it is desirable to give the file and other servers more bandwidth than clients, so they can be given two or more bytes per tick. This modified algorithm is called **weighted fair queueing** and is widely used. Sometimes the weight is equal to the number of virtual circuits or flows coming out of a machine, so each process gets equal bandwidth. An efficient implementation of the algorithm is discussed in (Shreedhar and Varghese, 1995).

Hop-by-Hop Choke Packets

At high speeds and over long distances, sending a choke packet to the source hosts does not work well because the reaction is so slow. Consider, for example, a host in San Francisco (router *A* in Fig. 5-30) that is sending traffic to a host in New York (router *D* in Fig. 5-30) at 155 Mbps. If the New York host begins to run out of buffers, it will take about 30 msec for a choke packet to get back to San Francisco to tell it to slow down. The choke packet propagation is shown as the second, third, and fourth steps in Fig. 5-30(a). In those 30 msec, another 4.6 megabits (e.g., over 10,000 ATM cells) will have been sent. Even if the host in San Francisco completely shuts down immediately, the 4.6 megabits in the pipe will continue to pour in and have to be dealt with. Only in the seventh diagram in Fig. 5-30(a) will the New York router notice a slower flow.

An alternative approach is to have the choke packet take effect at every hop it passes through, as shown in the sequence of Fig. 5-30(b). Here, as soon as the choke packet reaches *F*, *F* is required to reduce the flow to *D*. Doing so will require *F* to devote more buffers to the flow, since the source is still sending away at full blast, but it gives *D* immediate relief, like a headache remedy in a television commercial. In the next step, the choke packet reaches *E*, which tells *E* to reduce the flow to *F*. This action puts a greater demand on *E*'s buffers but gives *F* immediate relief. Finally, the choke packet reaches *A* and the flow genuinely slows down.

The net effect of this hop-by-hop scheme is to provide quick relief at the point of congestion at the price of using up more buffers upstream. In this way congestion can be nipped in the bud without losing any packets. The idea is discussed in more detail and simulation results are given in (Mishra and Kanakia, 1992).

5.3.7. Load Shedding

When none of the above methods make the congestion disappear, routers can bring out the heavy artillery: load shedding. **Load shedding** is a fancy way of saying that when routers are being inundated by packets that they cannot handle, they just throw them away. The term comes from the world of electrical power generation where it refers to the practice of utilities intentionally blacking out certain areas to save the entire grid from collapsing on hot summer days when the demand for electricity greatly exceeds the supply.

A router drowning in packets can just pick packets at random to drop, but usually it can do better than that. Which packet to discard may depend on the applications running. For file transfer, an old packet is worth more than a new one because dropping packet 6 and keeping packets 7 through 10 will cause a gap at the receiver that may force packets 6 through 10 to be retransmitted (if the receiver routinely discards out-of-order packets). In a 12-packet file, dropping 6 may require 7 through 12 to be retransmitted, whereas dropping 10 may require only 10 through 12 to be retransmitted. In contrast, for multimedia, a new packet is more important than an old one. The former policy (old is better than new) is often called **wine** and the latter (new is better than old) is often called **milk**.

A step above this in intelligence requires cooperation from the senders. For many applications, some packets are more important than others. For example, certain algorithms for compressing video periodically transmit an entire frame and then send subsequent frames as differences from the last full frame. In this case, dropping a packet that is part of a difference is preferable to dropping one that is part of a full frame. As another example, consider transmitting a document containing ASCII text and pictures. Losing a line of pixels in some image is far less damaging than losing a line of readable text.

To implement an intelligent discard policy, applications must mark their packets in priority classes to indicate how important they are. If they do this, when

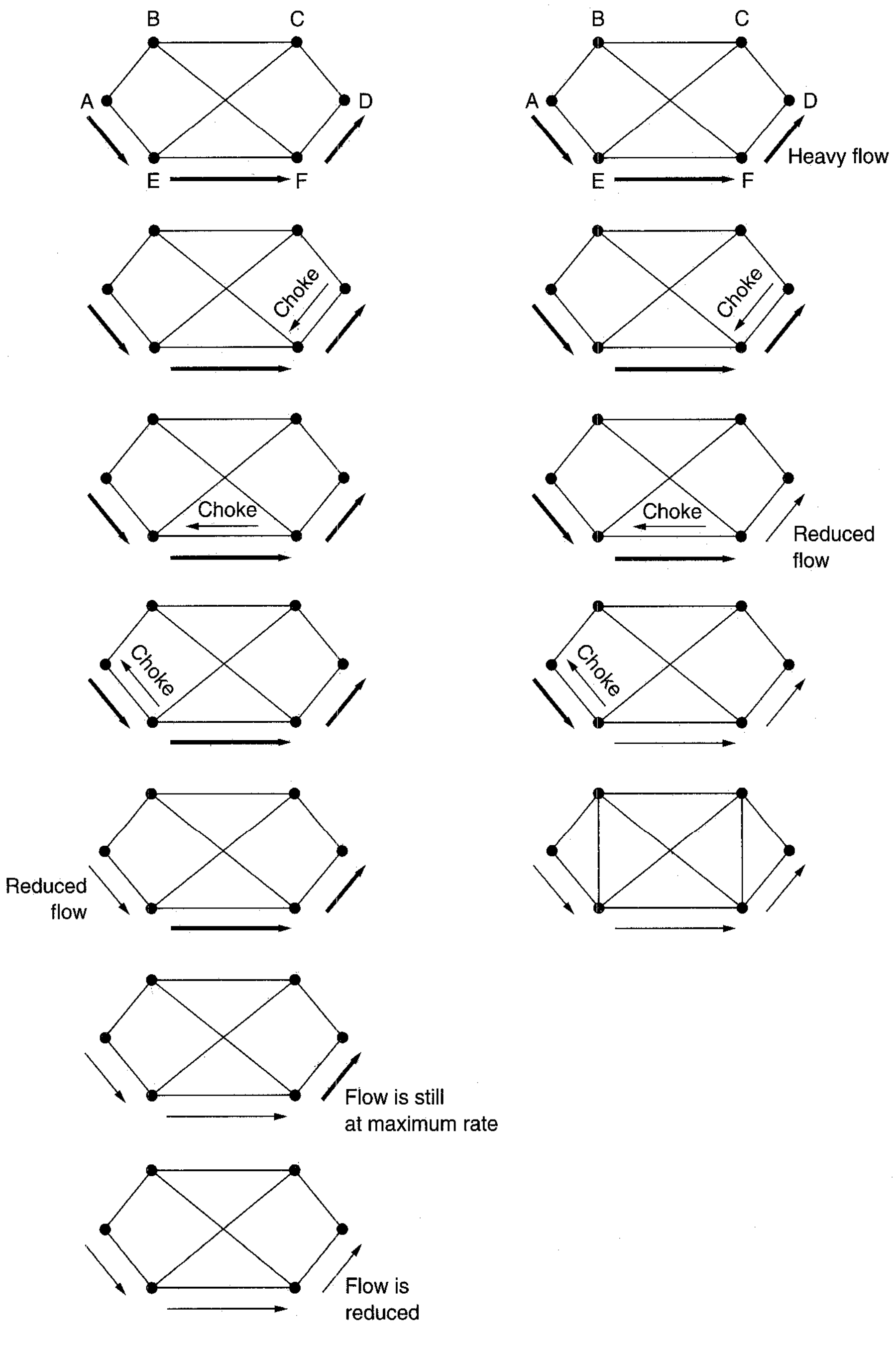


Fig. 5-30. (a) A choke packet that affects only the source. (b) A choke packet that affects each hop it passes through.

packets have to be discarded, routers can first drop packets from the lowest class, then the next lowest class, and so on. Of course, unless there is some significant incentive to mark packets as anything other than VERY IMPORTANT—NEVER, EVER DISCARD, nobody will do it.

The incentive might be in the form of money, with the low-priority packets being cheaper to send than the high-priority ones. Alternatively, priority classes could be coupled with traffic shaping. For example, there might be a rule saying that when the token bucket algorithm is being used and a packet arrives at a moment when no token is available, it may still be sent, provided that it is marked as the lowest possible priority, and thus subject to discard the instant trouble appears. Under conditions of light load, users might be happy to operate in this way, but as the load increases and packets actually begin to be discarded, they might cut back and only send packets when tokens are available.

Another option is to allow hosts to exceed the limits specified in the agreement negotiated when the virtual circuit was set up (e.g., use a higher bandwidth than allowed), but subject to the condition that all excess traffic be marked as low priority. Such a strategy is actually not a bad idea, because it makes more efficient use of idle resources, allowing hosts to use them as long as nobody else is interested, but without establishing a right to them when times get tough.

Marking packets by class requires one or more header bits in which to put the priority. ATM cells have 1 bit reserved in the header for this purpose, so every ATM cell is labeled either as low priority or high priority. ATM switches indeed use this bit when making discard decisions.

In some networks, packets are grouped together into larger units that are used for retransmission purposes. For example, in ATM networks, what we have been calling "packets" are fixed-length cells. These cells are just fragments of "messages." When a cell is dropped, ultimately the entire "message" will be retransmitted, not just the missing cell. Under these conditions, a router that drops a cell might as well drop all the rest of the cells in that message, since transmitting them costs bandwidth and wins nothing—even if they get through they will still be retransmitted later.

Simulation results show that when a router senses trouble on the horizon, it is better off starting to discard packets early, rather than wait until it becomes completely clogged up (Floyd and Jacobson, 1993; Romanow and Floyd, 1994). Doing so may prevent the congestion from getting a foothold.

5.3.8. Jitter Control

For applications such as audio and video transmission, it does not matter much if the packets take 20 msec or 30 msec to be delivered, as long as the transit time is constant. Having some packets taking 20 msec and others taking 30 msec will give an uneven quality to the sound or image. Thus the agreement might be that 99 percent of the packets be delivered with a delay in the range of 24.5 msec to 25.5 msec. The mean value chosen must be feasible, of course. In other words, an average amount of congestion must be calculated in.

The jitter can be bounded by computing the expected transit time for each hop along the path. When a packet arrives at a router, the router checks to see how much the packet is behind or ahead of its schedule. This information is stored in the packet and updated at each hop. If the packet is ahead of schedule, it is held just long enough to get it back on schedule. If it is behind schedule, the router tries to get it out the door quickly. In fact, the algorithm for determining which of several packets competing for an output line should go next can always choose the packet furthest behind in its schedule. In this way, packets that are ahead of schedule get slowed down and packets that are behind schedule get speeded up, in both cases reducing the amount of jitter.

5.3.9. Congestion Control for Multicasting

All of the congestion control algorithms discussed so far deal with messages from a single source to a single destination. In this section we will describe a way of managing multicast flows from multiple sources to multiple destinations. For example, imagine several closed-circuit television stations transmitting audio and video streams to a group of receivers, each of whom can view one or more stations at once and are free to switch from station to station at will. An application of this technology might be a video conference, in which each participant could focus on the current speaker or on the boss' expression, as desired.

In many multicast applications, groups can change membership dynamically, for example, as people enter a video conference or get bored and switch to a soap opera. Under these conditions, the approach of having the senders reserve bandwidth in advance does not work well, as it would require each sender to track all entries and exits of its audience and regenerate the spanning tree at each change. For a system designed to transmit cable television, with millions of subscribers, it would not work at all.

RSVP—Resource reSerVation Protocol

One interesting solution that can handle this environment is the **RSVP** protocol (Zhang et al., 1993). It allows multiple senders to transmit to multiple groups of receivers, permits individual receivers to switch channels freely, and optimizes bandwidth use while at the same time eliminating congestion.

In its simplest form, the protocol uses multicast routing using spanning trees, as discussed earlier. Each group is assigned a group address. To send to a group, a sender puts the group's address in its packets. The standard multicast routing algorithm then builds a spanning tree covering all group members. The routing algorithm is not part of RSVP. The only difference with normal multicasting is a little extra information that is multicast to the group periodically to tell the routers along the tree to maintain certain data structures in their memories.

As an example, consider the network of Fig. 5-31(a). Hosts 1 and 2 are multicast senders, and hosts 3, 4, and 5 are multicast receivers. In this example, the senders and receivers are disjoint, but in general, the two sets may overlap. The multicast trees for hosts 1 and 2 are shown in Fig. 5-31(b) and Fig. 5-31(c), respectively.

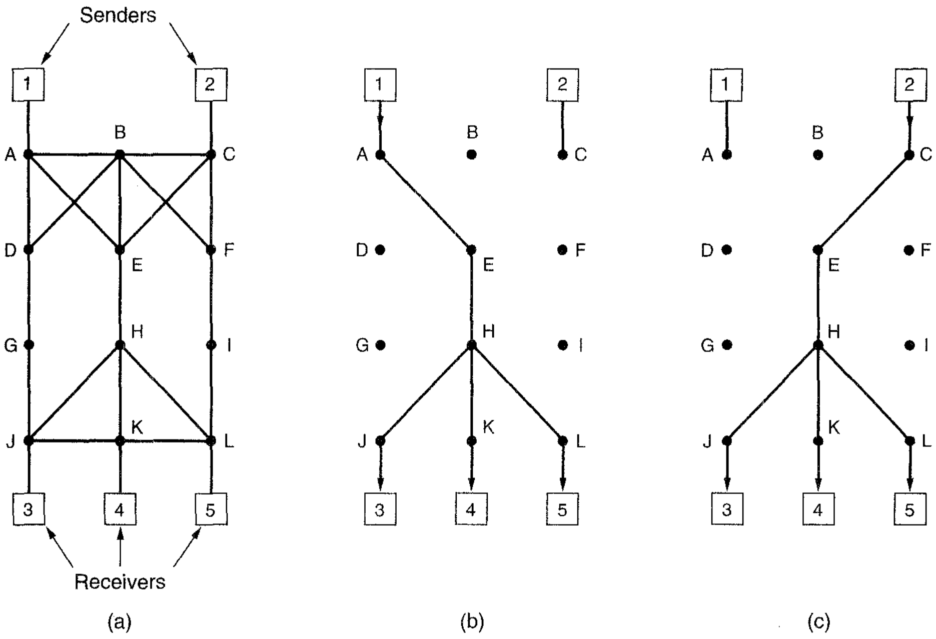


Fig. 5-31. (a) A network. (b) The multicast spanning tree for host 1. (c) The multicast spanning tree for host 2.

To get better reception and eliminate congestion, any of the receivers in a group can send a reservation message up the tree to the sender. The message is propagated using the reverse path forwarding algorithm discussed earlier. At each hop, the router notes the reservation and reserves the necessary bandwidth. If insufficient bandwidth is available, it reports back failure. By the time the message gets back to the source, bandwidth has been reserved all the way from the sender to the receiver making the reservation request along the spanning tree.

An example of such a reservation is shown in Fig. 5-32(a). Here host 3 has requested a channel to host 1. Once it has been established, packets can flow from 1 to 3 without congestion. Now consider what happens if host 3 next reserves a channel to the other sender, host 2, so the user can watch two television programs at once. A second path is reserved, as illustrated in Fig. 5-32(b). Note that two separate channels are needed from host 3 to router *E* because two independent streams are being transmitted.

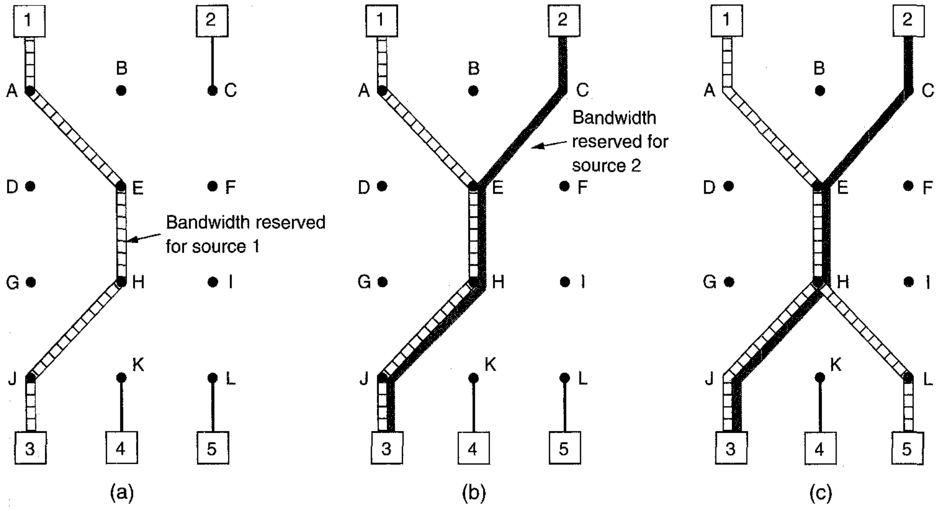

1
2
A
B
C
D
E
F
Bandwidth reserved for source 1
G
H
I
J
K
L
3
4
5
(a)
Bandwidth reserved for source 2
(b)
(c)

Fig. 5-32. (a) Host 3 requests a channel to host 1. (b) Host 3 then requests a second channel, to host 2. (c) Host 5 requests a channel to host 1.

Finally, in Fig. 5-32(c), host 5 decides to watch the program being transmitted by host 1 and also makes a reservation. First, dedicated bandwidth is reserved as far as router *H*. However, this router sees that it already has a feed from host 1, so if the necessary bandwidth has already been reserved, it does not have to reserve any more. Note that hosts 3 and 5 might have asked for different amounts of bandwidth (e.g., 3 has a black-and-white television set, so it does not want the color information), so the capacity reserved must be large enough to satisfy the greediest receiver.

When making a reservation, a receiver can (optionally) specify one or more sources that it wants to receive from. It can also specify whether these choices are fixed for the duration of the reservation, or whether the receiver wants to keep open the option of changing sources later. The routers use this information to optimize bandwidth planning. In particular, two receivers are only set up to share a path if they both agree not to change sources later on.

The reason for this strategy in the fully dynamic case is that reserved bandwidth is decoupled from the choice of source. Once a receiver has reserved bandwidth, it can switch to another source and keep that portion of the existing path that is valid for the new source. If host 2 is transmitting several video streams, for example, host 3 may switch between them at will without changing its reservation: the routers do not care what program the receiver is watching.

5.4. INTERNETWORKING

Up until now, we have implicitly assumed that there is a single homogeneous network, with each machine using the same protocol in each layer. Unfortunately, this assumption is wildly optimistic. Many different networks exist, including LANs, MANs, and WANs. Numerous protocols are in widespread use in every layer. In the following sections we will take a careful look at the issues that arise when two or more networks are together to form an **internet**.

Considerable controversy exists about the question of whether today's abundance of network types is a temporary condition that will go away as soon as everyone realizes how wonderful [fill in your favorite network] is, or whether it is an inevitable, but permanent feature of the world that is here to stay. Having different networks invariably means having different protocols.

We believe that a variety of different networks (and thus protocols) will always be around, for the following reasons. First of all, the installed base of different networks is large and growing. Nearly all UNIX shops run TCP/IP. Many large businesses still have mainframes running SNA. DEC is still developing DECnet. Personal computer LANs often use Novell NCP/IPX or AppleTalk. ATM systems are starting to be widespread. Finally, specialized protocols are often used on satellite, cellular, and infrared networks. This trend will continue for years due to the large number of existing networks and because not all vendors perceive it in their interest for their customers to be able to easily migrate to another vendor's system.

Second, as computers and networks get cheaper, the place where decisions get made moves downward. Many companies have a policy to the effect that purchases costing over a million dollars have to be approved by top management, purchases costing over 100,000 dollars have to be approved by middle management, but purchases under 100,000 dollars can be made by department heads without any higher approval. This can easily lead to the accounting department installing an Ethernet, the engineering department installing a token bus, and the personnel department installing a token ring.

Third, different networks (e.g., ATM and wireless) have radically different technology, so it should not be surprising that as new hardware developments occur, new software will be created to fit the new hardware. For example, the average home now is like the average office ten years ago: it is full of computers that do not talk to one another. In the future, it may be commonplace for the telephone, the television set, and other appliances all to be networked together, so they can be controlled remotely. This new technology will undoubtedly bring new protocols.

As an example of how different networks interact, consider the following example. At most universities, the computer science and electrical engineering departments have their own LANs, often different. In addition, the university computer center often has a mainframe and supercomputer, the former for faculty members in the humanities who do not wish to get into the computer maintenance business, and the latter for physicists who want to crunch numbers. As a consequence of these various networks and facilities, the following scenarios are easy to imagine:

1. LAN-LAN: A computer scientist downloading a file to engineering.
2. LAN-WAN: A computer scientist sending mail to a distant physicist.
3. WAN-WAN: Two poets exchanging sonnets.
4. LAN-WAN-LAN: Engineers at different universities communicating.

Figure 5-33 illustrates these four types of connections as dotted lines. In each case, it is necessary to insert a "black box" at the junction between two networks, to handle the necessary conversions as packets move from one network to the other.

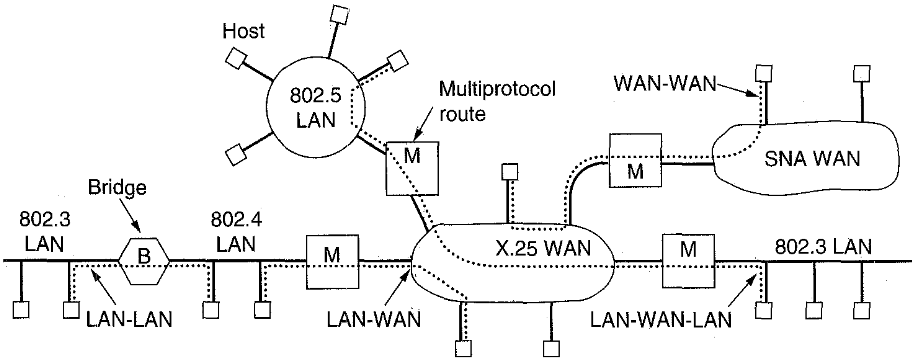


Fig. 5-33. Network interconnection.

The name used for the black box connecting two networks depends on the layer that does the work. Some common names are given below (although there is not much agreement on terminology in this area).

Layer 1: Repeaters copy individual bits between cable segments.

Layer 2: Bridges store and forward data link frames between LANs.

Layer 3: Multiprotocol routers forward packets between dissimilar networks.

Layer 4: Transport gateways connect byte streams in the transport layer.

Above 4: Application gateways allow interworking above layer 4.

For convenience, we will sometimes use the term "gateway" to mean any device that connects two or more dissimilar networks.

Repeaters are low-level devices that just amplify or regenerate weak signals. They are needed to provide current to drive long cables. In 802.3, for example, the timing properties of the MAC protocol (the value of τ chosen) allow cables up to 2.5 km, but the transceiver chips can only provide enough power to drive 500 meters. The solution is to use repeaters to extend the cable length where that is desired.

Unlike repeaters, which copy the bits as they arrive, **bridges** are store-and-forward devices. A bridge accepts an entire frame and passes it up to the data link layer where the checksum is verified. Then the frame is sent down to the physical layer for forwarding on a different network. Bridges can make minor changes to the frame before forwarding it, such as adding or deleting some fields from the frame header. Since they are data link layer devices, they do not deal with headers at layer 3 and above and cannot make changes or decisions that depend on them.

Multiprotocol routers are conceptually similar to bridges, except that they are found in the network layer. They just take incoming packets from one line and forward them on another, just as all routers do, but the lines may belong to different networks and use different protocols (e.g., IP, IPX, and the OSI connectionless packet protocol, CLNP). Like all routers, multiprotocol routers operate at the level of the network layer.

Transport gateways make a connection between two networks at the transport layer. We will discuss this possibility later when we come to concatenated virtual circuits.

Finally, **application gateways** connect two parts of an application in the application layer. For example, to send mail from an Internet machine using the Internet mail format to an ISO MOTIS mailbox, one could send the message to a mail gateway. The mail gateway would unpack the message, convert it to MOTIS format, and then forward it on the second network using the network and transport protocols used there.

When a gateway is between two WANs run by different organizations, possibly in different countries, the joint operation of one workstation-class machine can lead to a lot of finger pointing. To eliminate these problems, a slightly different approach can be taken. The gateway is effectively ripped apart in the middle and the two parts are connected with a wire. Each of the halves is called a **half-gateway** and each one is owned and operated by one of the network operators. The whole problem of gatewaying then reduces to agreeing to a common protocol to use on the wire, one that is neutral and does not favor either party. Figure 5-34 shows both full and half-gateways. Either kind can be used in any layer (e.g., half-bridges also exist).

That all said, the situation is murkier in practice than it is in theory. Many devices on the market combine bridge and router functionality. The key property of a pure bridge is that it examines data link layer frame headers and does not inspect or modify the network layer packets inside the frames. A bridge cannot

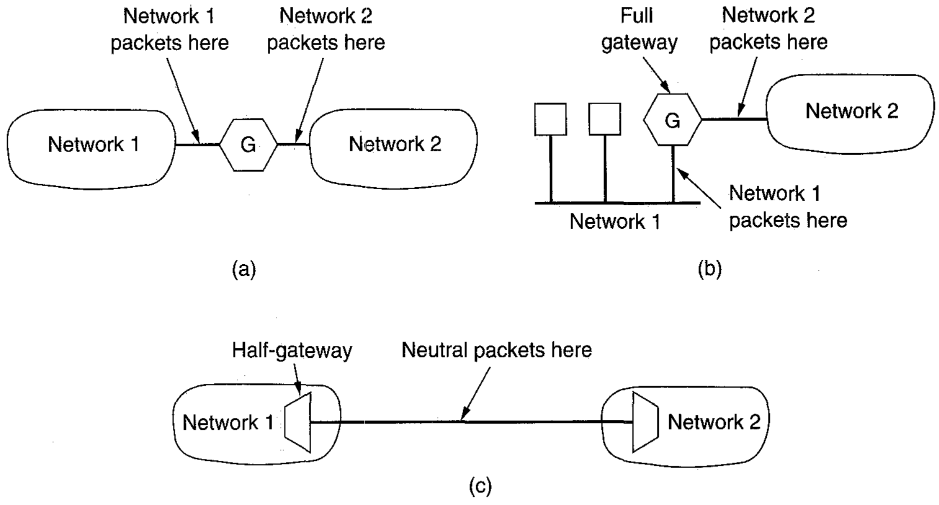


Fig. 5-34. (a) A full gateway between two WANs. (b) A full gateway between a LAN and a WAN. (c) Two half-gateways.

tell and does not care whether the frame it is forwarding from an 802.x LAN to an 802.y contains an IP, IPX, or CLNP packet in the payload field.

A router, in contrast, knows very well whether it is an IP router, an IPX router, a CLNP router, or all three combined. It examines these headers and makes decisions based on the addresses found there. On the other hand, when a pure router hands off a packet to the data link layer, it does not know or care whether it will be carried in an Ethernet frame or a token ring frame. That is the data link layer's responsibility.

The confusion in the industry comes from two sources. First, functionally, bridges and routers are not all that different. They each accept incoming PDUs (Protocol Data Units), examine some header fields, and make decisions about where to send the PDUs based on header information and internal tables.

Second, many commercial products are sold under the wrong label or combine the functionality of both bridges and routers. For example, source routing bridges are not really bridges at all, since they involve a protocol layer above the data link layer to do their job. For an illuminating discussion of bridges versus routers, see Chap. 12 of (Perlman, 1992).

5.4.1. How Networks Differ

Networks can differ in many ways. In Fig. 5-35 we list some of the differences that can occur in the network layer. It is papering over these differences that makes internetworking more difficult than operating within a single network.

Item	Some Possibilities
Service offered	Connection-oriented versus connectionless
Protocols	IP, IPX, CLNP, AppleTalk, DECnet, etc.
Addressing	Flat (802) versus hierarchical (IP)
Multicasting	Present or absent (also broadcasting)
Packet size	Every network has its own maximum
Quality of service	May be present or absent; many different kinds
Error handling	Reliable, ordered, and unordered delivery
Flow control	Sliding window, rate control, other, or none
Congestion control	Leaky bucket, choke packets, etc.
Security	Privacy rules, encryption, etc.
Parameters	Different timeouts, flow specifications, etc.
Accounting	By connect time, by packet, by byte, or not at all

Fig. 5-35. Some of the many ways networks can differ.

When packets sent by a source on one network must transit one or more foreign networks before reaching the destination network (which also may be different from the source network), many problems can occur at the interfaces between networks. To start with, when packets from a connection-oriented network must transit a connectionless one, they may be reordered, something the sender does not expect and the receiver is not prepared to deal with. Protocol conversions will often be needed, which can be difficult if the required functionality cannot be expressed. Address conversions will also be needed, which may require some kind of directory system. Passing multicast packets through a network that does not support multicasting requires generating separate packets for each destination.

The differing maximum packet sizes used by different networks is a major headache. How do you pass an 8000-byte packet through a network whose maximum size is 1500 bytes? Differing qualities of service is an issue when a packet that has real-time delivery constraints passes through a network that does offer any real-time guarantees.

Error, flow, and congestion control frequently differ among different networks. If the source and destination both expect all packets to be delivered in sequence without error, yet an intermediate network just discards packets whenever it smells congestion on the horizon, or packets can wander around aimlessly for a while and then suddenly emerge and be delivered, many applications will break. Different security mechanisms, parameter settings, and accounting rules, and even national privacy laws also can cause problems.

5.4.2. Concatenated Virtual Circuits

Two styles of internetworking are common: a connection-oriented concatenation of virtual circuit subnets, and a datagram internet style. We will now examine these in turn. In the concatenated virtual circuit model, shown in Fig. 5-36, a connection to a host in a distant network is set up in a way similar to the way connections are normally established. The subnet sees that the destination is remote and builds a virtual circuit to the router nearest the destination network. Then it constructs a virtual circuit from that router to an external "gateway" (multiprotocol router). This gateway records the existence of the virtual circuit in its tables and proceeds to build another virtual circuit to a router in the next subnet. This process continues until the destination host has been reached.

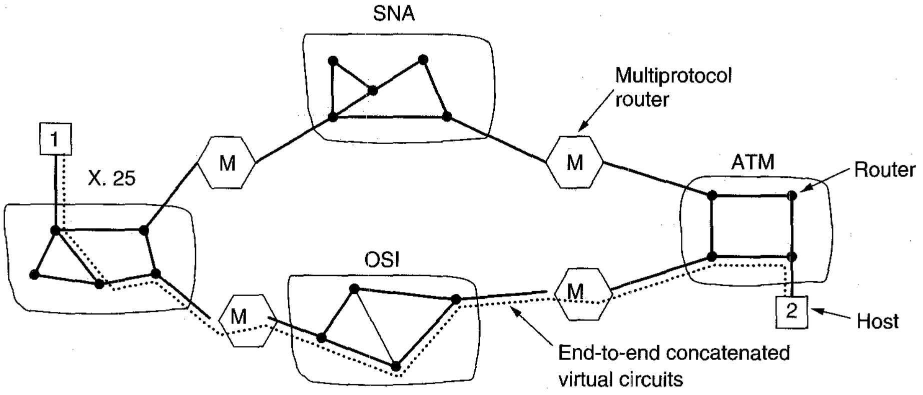


Fig. 5-36. Internetworking using concatenated virtual circuits.

Once data packets begin flowing along the path, each gateway relays incoming packets, converting between packet formats and virtual circuit numbers as needed. Clearly, all data packets must traverse the same sequence of gateways, and thus arrive in order.

The essential feature of this approach is that a sequence of virtual circuits is set up from the source through one or more gateways to the destination. Each gateway maintains tables telling which virtual circuits pass through it, where they are to be routed, and what the new virtual circuit number is.

Although Fig. 5-36 shows the connection made with a full gateway, it could equally well be done with half-gateways.

This scheme works best when all the networks have roughly the same properties. For example, if all of them guarantee reliable delivery of network layer packets, then barring a crash somewhere along the route, the flow from source to destination will also be reliable. Similarly, if none of them guarantee reliable delivery, then the concatenation of the virtual circuits is not reliable either. On the other hand, if the source machine is on a network that does guarantee reliable delivery, but one of the intermediate networks can lose packets, the concatenation has fundamentally changed the nature of the service.

Concatenated virtual circuits are also common in the transport layer. In particular, it is possible to build a bit pipe using, say, OSI, which terminates in a gateway, and have a TCP connection go from the gateway to the next gateway. In this manner, an end-to-end virtual circuit can be built spanning different networks and protocols.

5.4.3. Connectionless Internetworking

The alternative internetwork model is the datagram model, shown in Fig. 5-37. In this model, the only service the network layer offers to the transport layer is the ability to inject datagrams into the subnet and hope for the best. There is no notion of a virtual circuit at all in the network layer, let alone a concatenation of them. This model does not require all packets belonging to one connection to traverse the same sequence of gateways. In Fig. 5-37 datagrams from host 1 to host 2 are shown taking different routes through the internetwork. A routing decision is made separately for each packet, possibly depending on the traffic at the moment the packet is sent. This strategy can use multiple routes and thus achieve a higher bandwidth than the concatenated virtual circuit model. On the other hand, there is no guarantee that the packets arrive at the destination in order, assuming that they arrive at all.

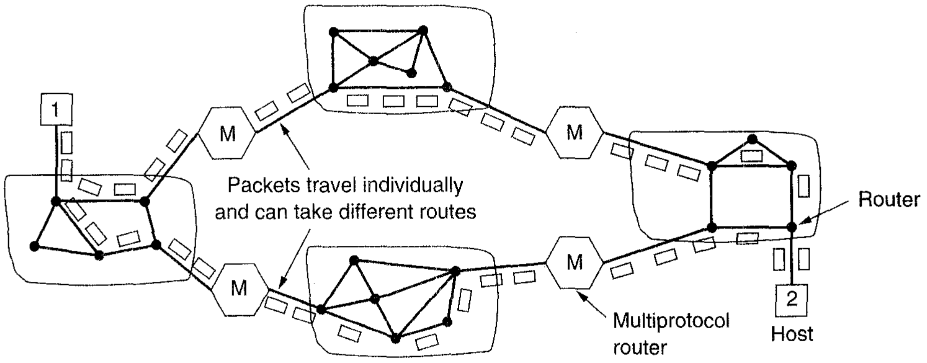


Fig. 5-37. A connectionless internet.

The model of Fig. 5-37 is not quite as simple as it looks. For one thing, if each network has its own network layer protocol, it is not possible for a packet from one network to transit another one. One could imagine the multiprotocol routers actually trying to translate from one format to another, but unless the two formats are close relatives with the same information fields, such conversions will always be incomplete and often doomed to failure. For this reason, conversion is rarely attempted.

A second, and more serious problem, is addressing. Imagine a simple case: a host on the Internet is trying to send an IP packet to a host on an adjoining OSI host. The OSI datagram protocol, CLNP, was based on IP and is close enough to it that a conversion might well work. The trouble is that IP packets all carry the 32-bit Internet address of the destination host in a header field. OSI hosts do not have 32-bit Internet addresses. They use decimal addresses similar to telephone numbers.

To make it possible for the multiprotocol router to convert between formats, someone would have to assign a 32-bit Internet address to each OSI host. Taken to the limit, this approach would mean assigning an Internet address to every machine in the world that an Internet host might want to talk to. It would also mean assigning an OSI address to every machine in the world that an OSI host might want to talk to. The same problem occurs with every other address space (SNA, AppleTalk, etc.). The problems here are insurmountable. In addition, someone would have to maintain a database mapping everything to everything.

Another idea is to design a universal "internet" packet and have all routers recognize it. This approach is, in fact, what IP is—a packet designed to be carried through many networks. The only problem is that IPX, CLNP, and other "universal" packets exist too, making all of them less than universal. Getting everybody to agree to a single format is just not possible.

Let us now briefly recap the two ways internetworking can be attacked. The concatenated virtual circuit model has essentially the same advantages as using virtual circuits within a single subnet: buffers can be reserved in advance, sequencing can be guaranteed, short headers can be used, and the troubles caused by delayed duplicate packets can be avoided.

It also has the same disadvantages: table space required in the routers for each open connection, no alternate routing to avoid congested areas, and vulnerability to router failures along the path. It also has the disadvantage of being difficult, if not impossible, to implement if one of the networks involved is an unreliable datagram network.

The properties of the datagram approach to internetworking are the same as those of datagram subnets: more potential for congestion, but also more potential for adapting to it, robustness in the face of router failures, and longer headers needed. Various adaptive routing algorithms are possible in an internet, just as they are within a single datagram network.

A major advantage of the datagram approach to internetworking is that it can be used over subnets that do not use virtual circuits inside. Many LANs, mobile networks (e.g., aircraft and naval fleets), and even some WANs fall into this category. When an internet includes one of these, serious problems occur if the internetworking strategy is based on virtual circuits.

5.4.4. Tunneling

Handling the general case of making two different networks interwork is exceedingly difficult. However, there is a common special case that is manageable. This case is where the source and destination hosts are on the same type of network, but there is a different network in between. As an example, think of an international bank with a TCP/IP based Ethernet in Paris, a TCP/IP based Ethernet in London, and a PTT WAN in between, as shown in Fig. 5-38.

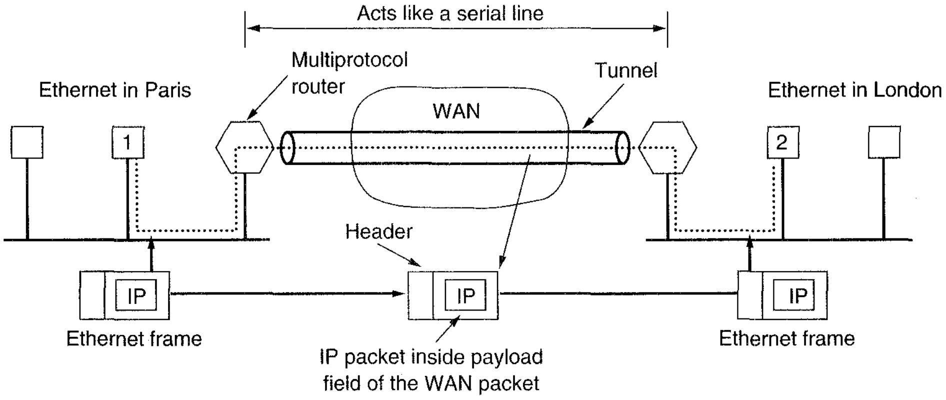


Fig. 5-38. Tunneling a packet from Paris to London.

The solution to this problem is a technique called **tunneling**. To send an IP packet to host 2, host 1 constructs the packet containing the IP address of host 2, inserts it into an Ethernet frame addressed to the Paris multiprotocol router, and puts it on the Ethernet. When the multiprotocol router gets the frame, it removes the IP packet, inserts it in the payload field of the WAN network layer packet, and addresses the latter to the WAN address of the London multiprotocol router. When it gets there, the London router removes the IP packet and sends it to host 2 inside an Ethernet frame.

The WAN can be seen as a big tunnel extending from one multiprotocol router to the other. The IP packet just travels from one end of the tunnel to the other, snug in its nice box. It does not have to worry about dealing with the WAN at all. Neither do the hosts on either Ethernet. Only the multiprotocol router has to understand IP and WAN packets. In effect, the entire distance from the middle of one multiprotocol router to the middle of the other acts like a serial line.

An analogy may make tunneling clearer. Consider a person driving her car from Paris to London. Within France, the car moves under its own power, but when it hits the English Channel, it is loaded into a high-speed train and transported to England through the Chunnel (cars are not permitted to drive through the Chunnel). Effectively, the car is being carried as freight, as depicted in Fig. 5-39.

At the far end, the car is let loose on the English roads and once again continues to move under its own power. Tunneling of packets through a foreign network works the same way.

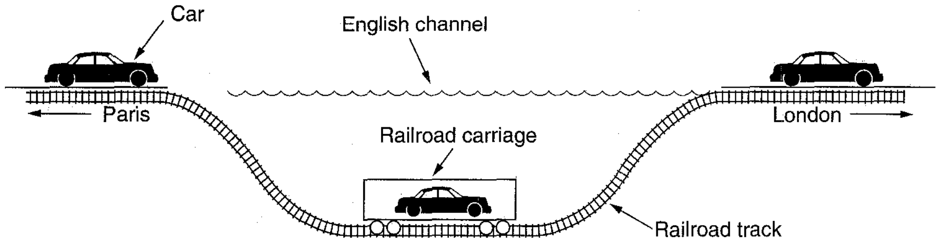


Fig. 5-39. Tunneling a car from France to England.

5.4.5. Internetwork Routing

Routing through an internetwork is similar to routing within a single subnet, but with some added complications. Consider, for example, the internetwork of Fig. 5-40(a) in which five networks are connected by six multiprotocol routers. Making a graph model of this situation is complicated by the fact that every multiprotocol router can directly access (i.e., send packets to) every other router connected to any network to which it is connected. For example, *B* in Fig. 5-40(a) can directly access *A* and *C* via network 2 and also *D* via network 3. This leads to the graph of Fig. 5-40(b).

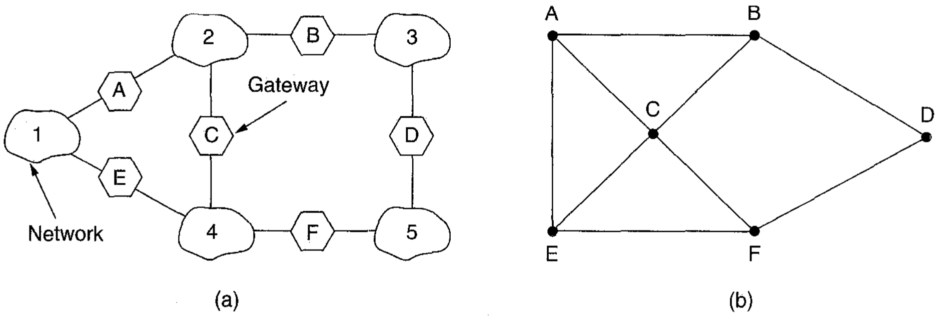


Fig. 5-40. (a) An internetwork. (b) A graph of the internetwork.

Once the graph has been constructed, known routing algorithms, such as the distance vector and link state algorithms, can be applied to the set of multiprotocol routers. This gives a two-level routing algorithm: within each network an **interior gateway protocol** is used, but between the networks, an **exterior gateway protocol** is used ("gateway" is an older term for "router"). In fact, since each network is independent, they may all use different algorithms. Because each network in an internetwork is independent of all the others, it is often referred to as an **Autonomous System** (**AS**).

A typical internet packet starts out on its LAN addressed to the local multiprotocol router (in the MAC layer header). After it gets there, the network layer code decides which multiprotocol router to forward the packet to, using its own routing tables. If that router can be reached using the packet's native network protocol, it is forwarded there directly. Otherwise it is tunneled there, encapsulated in the protocol required by the intervening network. This process is repeated until the packet reaches the destination network.

One of the differences between internetwork routing and intranetwork routing is that internetwork routing often requires crossing international boundaries. Various laws suddenly come into play, such as Sweden's strict privacy laws about exporting personal data about Swedish citizens from Sweden. Another example is the Canadian law saying that data traffic originating in Canada and ending in Canada may not leave the country. This law means that traffic from Windsor, Ontario to Vancouver may not be routed via nearby Detroit.

Another difference between interior and exterior routing is the cost. Within a single network, a single charging algorithm normally applies. However, different networks may be under different managements, and one route may be less expensive than another. Similarly, the quality of service offered by different networks may be different, and this may be a reason to choose one route over another.

In a large internetwork, choosing the best route may be a time-consuming operation. Estrin et al. (1992) have proposed dealing with this problem by precomputing routes for popular (source, destination) pairs and storing them in a database to be consulted at route selection time.

5.4.6. Fragmentation

Each network imposes some maximum size on its packets. These limits have various causes, among them:

1. Hardware (e.g., the width of a TDM transmission slot).
2. Operating system (e.g., all buffers are 512 bytes).
3. Protocols (e.g., the number of bits in the packet length field).
4. Compliance with some (inter)national standard.
5. Desire to reduce error induced retransmissions to some level.
6. Desire to prevent one packet from occupying the channel too long.

The result of all these factors is that the network designers are not free to choose any maximum packet size they wish. Maximum payloads range from 48 bytes

(ATM cells) to 65,515 bytes (IP packets), although the payload size in higher layers is often larger.

An obvious problem appears when a large packet wants to travel through a network whose maximum packet size is too small. One solution is to make sure the problem does not occur in the first place. In other words, the internet should use a routing algorithm that avoids sending packets through networks that cannot handle them. However, this solution is no solution at all. What happens if the original source packet is too large to be handled by the destination network? The routing algorithm can hardly bypass the destination.

Basically, the only solution to the problem is to allow gateways to break packets up into **fragments**, sending each fragment as a separate internet packet. However, as every parent of a small child knows, converting a large object into small fragments is considerably easier than the reverse process. (Physicists have even given this effect a name: the second law of thermodynamics.) Packet-switching networks, too, have trouble putting the fragments back together again.

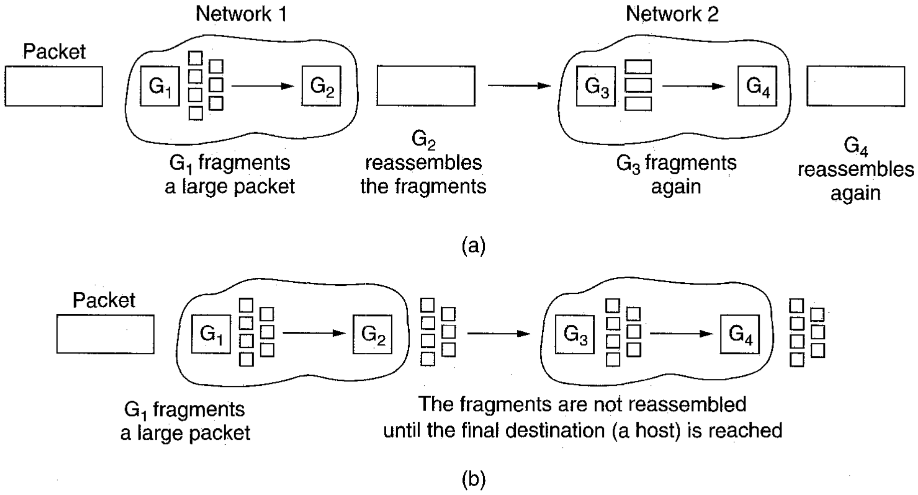

Network 1
Network 2
Packet
G1
G2
G3
G4
G1 fragments a large packet
G2 reassembles the fragments
G3 fragments again
G4 reassembles again
(a)
Packet
G1
G2
G3
G4
G1 fragments a large packet
The fragments are not reassembled until the final destination (a host) is reached
(b)

Fig. 5-41. (a) Transparent fragmentation. (b) Nontransparent fragmentation.

Two opposing strategies exist for recombining the fragments back into the original packet. The first strategy is to make fragmentation caused by a "small-packet" network transparent to any subsequent networks through which the packet must pass on its way to the ultimate destination. This option is shown in Fig. 5-41(a). When an oversized packet arrives at a gateway, the gateway breaks it up into fragments. Each fragment is addressed to the same exit gateway, where the pieces are recombined. In this way passage through the small-packet network has been made transparent. Subsequent networks are not even aware that fragmentation has occurred. ATM networks, for example, have special hardware to provide transparent fragmentation of packets into cells and then reassembly of cells into packets. In the ATM world, fragmentation is called segmentation; the concept is the same, but some of the details are different.

Transparent fragmentation is simple but has some problems. For one thing, the exit gateway must know when it has received all the pieces, so that either a count field or an "end of packet" bit must be included in each packet. For another thing, all packets must exit via the same gateway. By not allowing some fragments to follow one route to the ultimate destination, and other fragments a disjoint route, some performance may be lost. A last problem is the overhead required to repeatedly reassemble and then refragment a large packet passing through a series of small-packet networks.

The other fragmentation strategy is to refrain from recombining fragments at any intermediate gateways. Once a packet has been fragmented, each fragment is treated as though it were an original packet. All fragments are passed through the exit gateway (or gateways), as shown in Fig. 5-41(b). Recombination occurs only at the destination host.

Nontransparent fragmentation also has some problems. For example, it requires *every* host to be able to do reassembly. Yet another problem is that when a large packet is fragmented the total overhead increases, because each fragment must have a header. Whereas in the first method this overhead disappears as soon as the small-packet network is exited, in this method the overhead remains for the rest of the journey. An advantage of this method, however, is that multiple exit gateways can now be used and higher performance can be achieved. Of course, if the concatenated virtual circuit model is being used, this advantage is of no use.

When a packet is fragmented, the fragments must be numbered in such a way that the original data stream can be reconstructed. One way of numbering the fragments is to use a tree. If packet 0 must be split up, the pieces are called 0.0, 0.1, 0.2, etc. If these fragments themselves must be fragmented later on, the pieces are numbered 0.0.0, 0.0.1, 0.0.2, . . . , 0.1.0, 0.1.1, 0.1.2, etc. If enough fields have been reserved in the header for the worst case and no duplicates are generated anywhere, this scheme is sufficient to ensure that all the pieces can be correctly reassembled at the destination, no matter what order they arrive in.

However, if even one network loses or discards packets, there is a need for end-to-end retransmissions, with unfortunate effects for the numbering system. Suppose that a 1024-bit packet is initially fragmented into four equal-sized fragments, 0.0, 0.1, 0.2, and 0.3. Fragment 0.1 is lost, but the other parts arrive at the destination. Eventually, the source times out and retransmits the original packet again. Only this time the route taken passes through a network with a 512-bit limit, so two fragments are generated. When the new fragment 0.1 arrives at the destination, the receiver will think that all four pieces are now accounted for and reconstruct the packet incorrectly.

A completely different (and better) numbering system is for the internetwork protocol to define an elementary fragment size small enough that the elementary fragment can pass through every network. When a packet is fragmented, all the pieces are equal to the elementary fragment size except the last one, which may be shorter. An internet packet may contain several fragments, for efficiency reasons. The internet header must provide the original packet number, and the number of the (first) elementary fragment contained in the packet. As usual, there must also be a bit indicating that the last elementary fragment contained within the internet packet is the last one of the original packet.

This approach requires two sequence fields in the internet header: the original packet number, and the fragment number. There is clearly a trade-off between the size of the elementary fragment and the number of bits in the fragment number. Because the elementary fragment size is presumed to be acceptable to every network, subsequent fragmentation of an internet packet containing several fragments causes no problem. The ultimate limit here is to have the elementary fragment be a single bit or byte, with the fragment number then being the bit or byte offset within the original packet, as shown in Fig. 5-42.

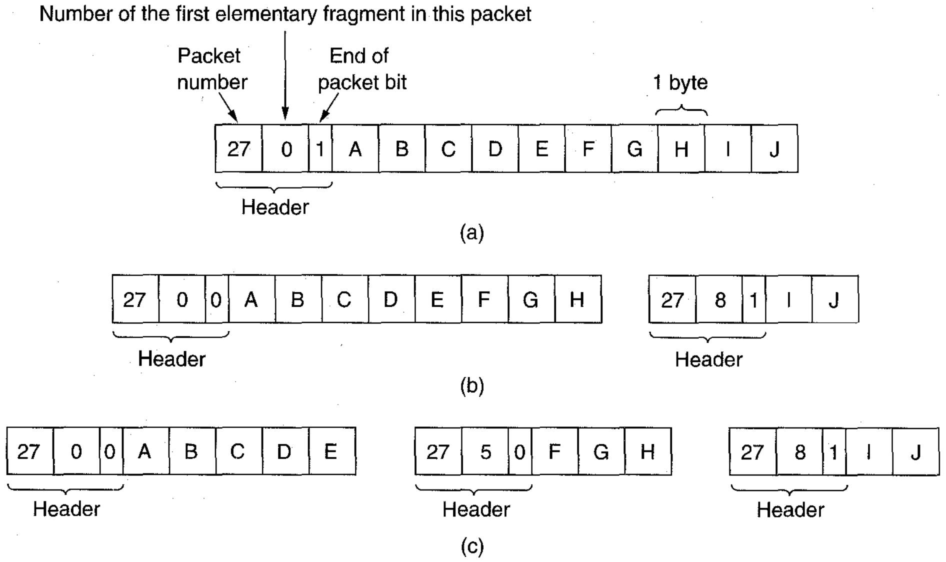


Fig. 5-42. Fragmentation when the elementary data size is 1 byte. (a) Original packet, containing 10 data bytes. (b) Fragments after passing through a network with maximum packet size of 8 bytes. (c) Fragments after passing through a size 5 gateway.

Some internet protocols take this method even further and consider the entire transmission on a virtual circuit to be one giant packet, so that each fragment contains the absolute byte number of the first byte within the fragment. Some other issues relating to fragmentation are discussed in (Kent and Mogul, 1987).

5.4.7. Firewalls

The ability to connect any computer, anywhere, to any other computer, anywhere, is a mixed blessing. For individuals at home, wandering around the Internet is lots of fun. For corporate security managers, it is a nightmare. Most companies have large amounts of confidential information on-line—trade secrets, product development plans, marketing strategies, financial analyses, etc. Disclosure of this information to a competitor could have dire consequences.

In addition to the danger of information leaking out, there is also a danger of information leaking in. In particular, viruses, worms, and other digital pests (Kaufman et al., 1995) can breach security, destroy valuable data, and waste large amounts of administrators' time trying to clean up the mess they leave. Often they are imported by careless employees who want to play some nifty new game.

Consequently, mechanisms are needed to keep "good" bits in and "bad" bits out. One method is to use encryption. This approach protects data in transit between secure sites. We will study it in Chap. 7. However, encryption does nothing to keep digital pests and hackers out. To accomplish this goal, we need to look at firewalls (Chapman and Zwicky, 1995; and Cheswick and Bellovin, 1994).

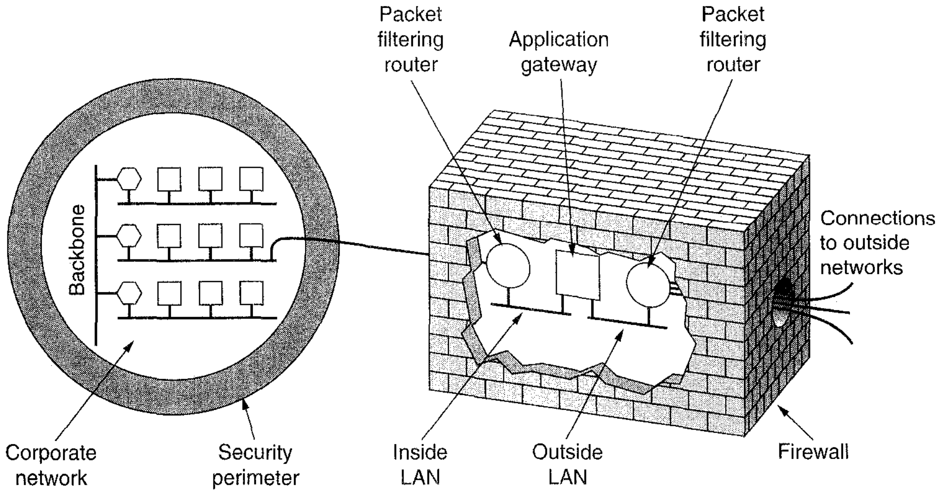


Fig. 5-43. A firewall consisting of two packet filters and an application gateway.

Firewalls are just a modern adaptation of that old medieval security standby: digging a deep moat around your castle. This design forced everyone entering or leaving the castle to pass over a single drawbridge, where they could be inspected by the I/O police. With networks, the same trick is possible: a company can have many LANs connected in arbitrary ways, but all traffic to or from the company is forced through an electronic drawbridge (firewall), as shown in Fig. 5-43.

The firewall in this configuration has two components: two routers that do packet filtering and an application gateway. Simpler configurations also exist, but the advantage of this design is that every packet must transit two filters and an application gateway to go in or out. No other route exists. Readers who think that one security checkpoint is enough clearly have not made an international flight on a scheduled airline recently.

Each **packet filter** is a standard router equipped with some extra functionality. The extra functionality allows every incoming or outgoing packet to be inspected. Packets meeting some criterion are forwarded normally. Those that fail the test are dropped.

In Fig. 5-43, most likely the packet filter on the inside LAN checks outgoing packets and the one on the outside LAN checks incoming packets. Packets crossing the first hurdle go to the application gateway for further examination. The point of putting the two packet filters on different LANs is to ensure that no packet gets in or out without having to pass through the application gateway: there is no path around it.

Packet filters are typically driven by tables configured by the system administrator. These tables list sources and destinations that are acceptable, sources and destinations that are blocked, and default rules about what to do with packets coming from or going to other machines.

In the common case of a UNIX setting, a source or destination consists of an IP address and a port. Ports indicate which service is desired. For example, port 23 is for Telnet, port 79 is for Finger, and port 119 is for USENET news. A company could block incoming packets for all IP addresses combined with one of these ports. In this way, no one outside the company could log in via Telnet, or look up people using the Finger daemon. Furthermore, the company would be spared from having employees spend all day reading USENET news.

Blocking outgoing packets is trickier because although most sites stick to the standard port naming conventions, they are not forced to do so. Furthermore, for some important services, such as FTP (File Transfer Protocol), port numbers are assigned dynamically. In addition, although blocking TCP connections is difficult, blocking UDP packets is even harder because so little is known a priori about what they will do. Many packet filters simply ban UDP traffic altogether.

The second half of the firewall mechanism is the **application gateway**. Rather than just looking at raw packets, the gateway operates at the application level. A mail gateway, for example, can be set up to examine each message going in or coming out. For each one it makes a decision to transmit or discard it based on header fields, message size, or even the content (e.g., at a military installation, the presence of words like "nuclear" or "bomb" might cause some special action to be taken).

Installations are free to set up one or more application gateways for specific applications, but it is not uncommon for suspicious organizations to permit email in and out, and perhaps use of the World Wide Web, but ban everything else as too dicey. Combined with encryption and packet filtering, this arrangement offers a limited amount of security at the cost of some inconvenience.

One final note concerns wireless communication and firewalls. It is easy to design a system that is logically completely secure, but which, in practice, leaks like a sieve. This situation can occur if some of the machines are wireless and use radio communication, which passes right over the firewall in both directions.

5.5. THE NETWORK LAYER IN THE INTERNET

At the network layer, the Internet can be viewed as a collection of subnetworks or **Autonomous Systems** (**ASes**) that are connected together. There is no real structure, but several major backbones exist. These are constructed from high-bandwidth lines and fast routers. Attached to the backbones are regional (midlevel) networks, and attached to these regional networks are the LANs at many universities, companies, and Internet service providers. A sketch of this quasihierarchical organization is given in Fig. 5-44.

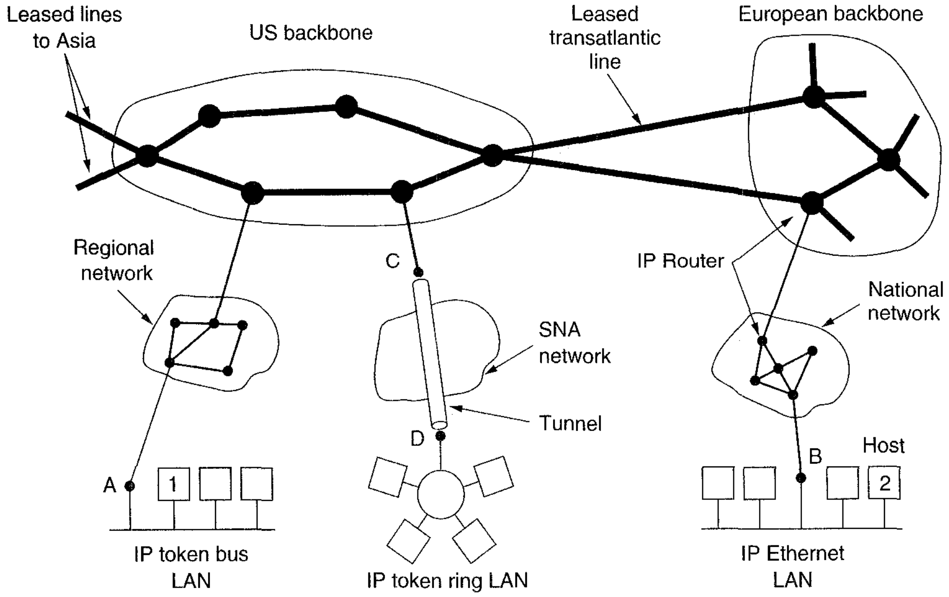


Fig. 5-44. The Internet is an interconnected collection of many networks.

The glue that holds the Internet together is the network layer protocol, **IP** (**Internet Protocol**). Unlike most older network layer protocols, it was designed from the beginning with internetworking in mind. A good way to think of the network layer is this. Its job is to provide a best-efforts way to transport datagrams from source to destination, without regard to whether or not these machines are on the same network, or whether or not there are other networks in between them.

Communication in the Internet works as follows. The transport layer takes data streams and breaks them up into datagrams. In theory, datagrams can be up to 64 Kbytes each, but in practice they are usually around 1500 bytes. Each datagram is transmitted through the Internet, possibly being fragmented into smaller units as it goes. When all the pieces finally get to the destination machine, they are reassembled by the network layer into the original datagram. This datagram is then handed to the transport layer, which inserts it into the receiving process' input stream.

5.5.1. The IP Protocol

An appropriate place to start our study of the network layer in the Internet is the format of the IP datagrams themselves. An IP datagram consists of a header part and a text part. The header has a 20-byte fixed part and a variable length optional part. The header format is shown in Fig. 5-45. It is transmitted in big endian order: from left to right, with the high-order bit of the *Version* field going first. (The SPARC is big endian; the Pentium is little endian.) On little endian machines, software conversion is required on both transmission and reception.

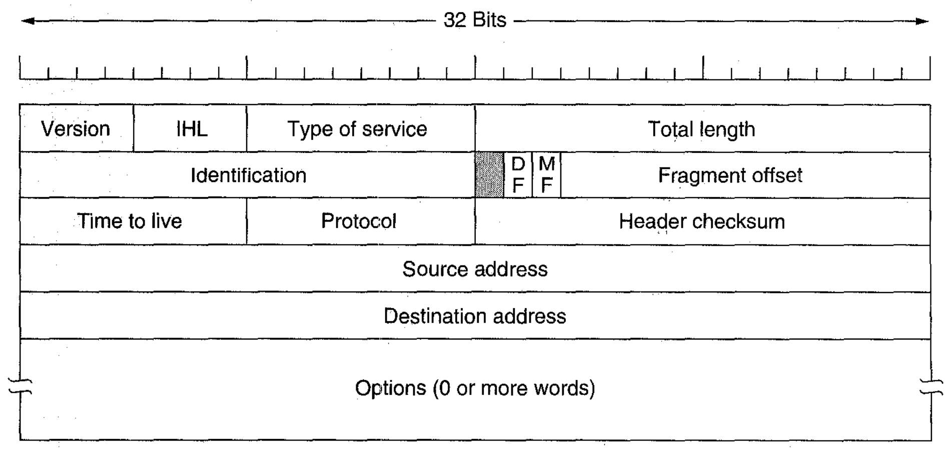


Fig. 5-45. The IP (Internet Protocol) header.

The *Version* field keeps track of which version of the protocol the datagram belongs to. By including the version in each datagram, it becomes possible to have the transition between versions take months, or even years, with some machines running the old version and others running the new one.

Since the header length is not constant, a field in the header, *IHL*, is provided to tell how long the header is, in 32-bit words. The minimum value is 5, which applies when no options are present. The maximum value of this 4-bit field is 15, which limits the header to 60 bytes, and thus the options field to 40 bytes. For some options, such as one that records the route a packet has taken, 40 bytes is far too small, making the option useless.

The *Type of service* field allows the host to tell the subnet what kind of service it wants. Various combinations of reliability and speed are possible. For digitized voice, fast delivery beats accurate delivery. For file transfer, error-free transmission is more important than fast transmission.

The field itself contains (from left to right), a three-bit *Precedence* field, three flags, *D*, *T*, and *R*, and 2 unused bits. The *Precedence* field is a priority, from 0 (normal) to 7 (network control packet). The three flag bits allow the host to specify what it cares most about from the set {Delay, Throughput, Reliability}. In theory, these fields allow routers to make choices between, for example, a satellite link with high throughput and high delay or a leased line with low throughput and low delay. In practice, current routers ignore the *Type of Service* field altogether.

The *Total length* includes everything in the datagram—both header and data. The maximum length is 65,535 bytes. At present, this upper limit is tolerable, but with future gigabit networks larger datagrams will be needed.

The *Identification* field is needed to allow the destination host to determine which datagram a newly arrived fragment belongs to. All the fragments of a datagram contain the same *Identification* value.

Next comes an unused bit and then two 1-bit fields. *DF* stands for Don't Fragment. It is an order to the routers not to fragment the datagram because the destination is incapable of putting the pieces back together again. For example, when a computer boots, its ROM might ask for a memory image to be sent to it as a single datagram. By marking the datagram with the *DF* bit, the sender knows it will arrive in one piece, even if this means that the datagram must avoid a small-packet network on the best path and take a suboptimal route. All machines are required to accept fragments of 576 bytes or less.

MF stands for More Fragments. All fragments except the last one have this bit set. It is needed to know when all fragments of a datagram have arrived.

The *Fragment offset* tells where in the current datagram this fragment belongs. All fragments except the last one in a datagram must be a multiple of 8 bytes, the elementary fragment unit. Since 13 bits are provided, there is a maximum of 8192 fragments per datagram, giving a maximum datagram length of 65,536 bytes, one more than the *Total length* field.

The *Time to live* field is a counter used to limit packet lifetimes. It is supposed to count time in seconds, allowing a maximum lifetime of 255 sec. It must be decremented on each hop and is supposed to be decremented multiple times when queued for a long time in a router. In practice, it just counts hops. When it hits zero, the packet is discarded and a warning packet is sent back to the source host. This feature prevents datagrams for wandering around forever, something that otherwise might happen if the routing tables ever become corrupted.

When the network layer has assembled a complete datagram, it needs to know what to do with it. The *Protocol* field tells it which transport process to give it to. TCP is one possibility, but so are UDP and some others. The numbering of protocols is global across the entire Internet and is defined in RFC 1700.

The *Header checksum* verifies the header only. Such a checksum is useful for detecting errors generated by bad memory words inside a router. The algorithm is to add up all the 16-bit halfwords as they arrive, using one's complement arithmetic and then take the one's complement of the result. For purposes of this algorithm, the *Header checksum* is assumed to be zero upon arrival. This algorithm is more robust than using a normal add. Note that the *Header checksum* must be recomputed at each hop, because at least one field always changes (the *Time to live* field), but tricks can be used to speed up the computation.

The *Source address* and *Destination address* indicate the network number and host number. We will discuss Internet addresses in the next section. The *Options* field was designed to provide an escape to allow subsequent versions of the protocol to include information not present in the original design, to permit experimenters to try out new ideas, and to avoid allocating header bits to information that is rarely needed. The options are variable length. Each begins with a 1-byte code identifying the option. Some options are followed by a 1-byte option length field, and then one or more data bytes. The *Options* field is padded out to a multiple of four bytes. Currently five options are defined, as listed in Fig. 5-46, but not all routers support all of them.

Option	Description
Security	Specifies how secret the datagram is
Strict source routing	Gives the complete path to be followed
Loose source routing	Gives a list of routers not to be missed
Record route	Makes each router append its IP address
Timestamp	Makes each router append its address and timestamp

Fig. 5-46. IP options.

The *Security* option tells how secret the information is. In theory, a military router might use this field to specify not to route through certain countries the military considers to be "bad guys." In practice, all routers ignore it, so its only practical function is to help spies find the good stuff more easily.

The *Strict source routing* option gives the complete path from source to destination as a sequence of IP addresses. The datagram is required to follow that exact route. It is most useful for system managers to send emergency packets when the routing tables are corrupted, or for making timing measurements.

The *Loose source routing* option requires the packet to traverse the list of routers specified, and in the order specified, but it is allowed to pass through other routers on the way. Normally, this option would only provide a few routers, to force a particular path. For example, to force a packet from London to Sydney to go west instead of east, this option might specify routers in New York, Los Angeles, and Honolulu. This option is most useful when political or economic considerations dictate passing through or avoiding certain countries.

The *Record route* option tells the routers along the path to append their IP address to the option field. This allows system managers to track down bugs in the routing algorithms ("Why are packets from Houston to Dallas all visiting Tokyo first?") When the ARPANET was first set up, no packet ever passed through more than nine routers, so 40 bytes of option was ample. As mentioned above, now it is too small.

Finally, the *Timestamp* option is like the *Record route* option, except that in addition to recording its 32-bit IP address, each router also records a 32-bit timestamp. This option, too, is mostly for debugging routing algorithms.

5.5.2. IP Addresses

Every host and router on the Internet has an IP address, which encodes its network number and host number. The combination is unique: no two machines have the same IP address. All IP addresses are 32 bits long and are used in the *Source address* and *Destination address* fields of IP packets. The formats used for IP address are shown in Fig. 5-47. Those machines connected to multiple networks have a different IP address on each network.

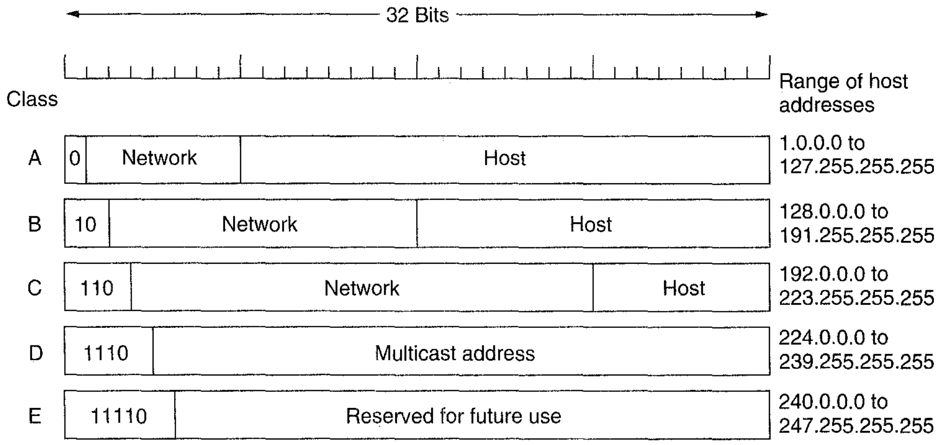

32 Bits
Class
Range of host addresses
A
0
Network
Host
1.0.0.0 to 127.255.255.255
B
10
Network
Host
128.0.0.0 to 191.255.255.255
C
110
Network
Host
192.0.0.0 to 223.255.255.255
D
1110
Multicast address
224.0.0.0 to 239.255.255.255
E
11110
Reserved for future use
240.0.0.0 to 247.255.255.255

Fig. 5-47. IP address formats.

The class A, B, C, and D formats allow for up to 126 networks with 16 million hosts each, 16,382 networks with up to 64K hosts, 2 million networks, (e.g., LANs), with up to 254 hosts each, and multicast, in which a datagram is directed to multiple hosts. Addresses beginning with 11110 are reserved for future use. Tens of thousands of networks are now connected to the Internet, and the number doubles every year. Network numbers are assigned by the **NIC** (**Network Information Center**) to avoid conflicts.

Network addresses, which are 32-bit numbers, are usually written in **dotted decimal notation**. In this format, each of the 4 bytes is written in decimal, from 0 to 255. For example, the hexadecimal address C0290614 is written as 192.41.6.20. The lowest IP address is 0.0.0.0 and the highest is 255.255.255.255.

The values 0 and −1 have special meanings, as shown in Fig. 5-48. The value 0 means this network or this host. The value of −1 is used as a broadcast address to mean all hosts on the indicated network.

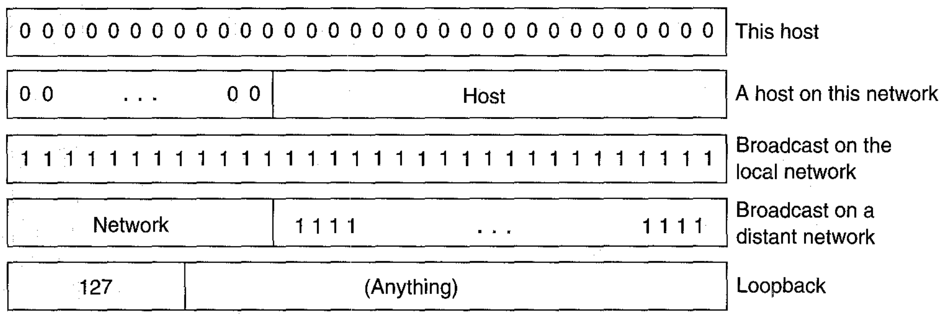


Fig. 5-48. Special IP addresses.

The IP address 0.0.0.0 is used by hosts when they are being booted but is not used afterward. IP addresses with 0 as network number refer to the current network. These addresses allow machines to refer to their own network without knowing its number (but they have to know its class to know how many 0s to include). The address consisting of all 1s allows broadcasting on the local network, typically a LAN. The addresses with a proper network number and all 1s in the host field allow machines to send broadcast packets to distant LANs anywhere in the Internet. Finally, all addresses of the form 127.xx.yy.zz are reserved for loopback testing. Packets sent to that address are not put out onto the wire; they are processed locally and treated as incoming packets. This allows packets to be sent to the local network without the sender knowing its number. This feature is also used for debugging network software.

5.5.3. Subnets

As we have seen, all the hosts in a network must have the same network number. This property of IP addressing can cause problems as networks grow. For example, consider a company that starts out with one class C LAN on the

Internet. As time goes on, it might acquire more than 254 machines, and thus need a second class C address. Alternatively, it might acquire a second LAN of a different type and want a separate IP address for it (the LANs could be bridged to form a single IP network, but bridges have their own problems). Eventually, it might end up with many LANs, each with its own router and each with its own class C network number.

As the number of distinct local networks grows, managing them can become a serious headache. Every time a new network is installed the system administrator has to contact NIC to get a new network number. Then this number must be announced worldwide. Furthermore, moving a machine from one LAN to another requires it to change its IP address, which in turn may mean modifying its configuration files and also announcing the new IP address to the world. If some other machine is given the newly-released IP address, that machine will get email and other data intended for the original machine until the address has propagated all over the world.

The solution to these problems is to allow a network to be split into several parts for internal use but still act like a single network to the outside world. In the Internet literature, these parts are called **subnets**. As we mentioned in Chap. 1, this usage conflicts with "subnet" to mean the set of all routers and communication lines in a network. Hopefully it will be clear from the context which meaning is intended. In this section, the new definition will be the one used. If our growing company started up with a class B address instead of a class C address, it could start out just numbering the hosts from 1 to 254. When the second LAN arrived, it could decide, for example, to split the 16-bit host number into a 6-bit subnet number and a 10-bit host number, as shown in Fig. 5-49. This split allows 62 LANs (0 and −1 are reserved), each with up to 1022 hosts.

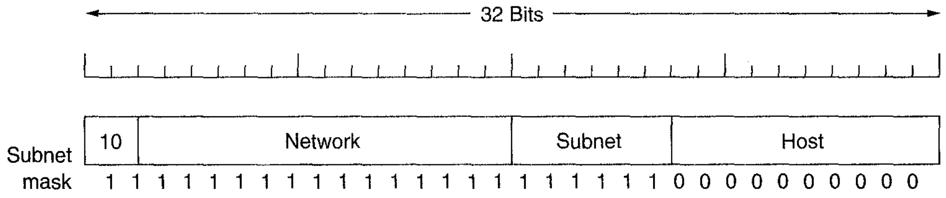


Fig. 5-49. One of the ways to subnet a class B network.

Outside the network, the subnetting is not visible, so allocating a new subnet does not require contacting NIC or changing any external databases. In this example, the first subnet might use IP addresses starting at 130.50.4.1, the second subnet might start at 130.50.8.1, and so on.

To see how subnets work, it is necessary to explain how IP packets are processed at a router. Each router has a table listing some number of (network, 0) IP addresses and some number of (this-network, host) IP addresses. The first kind tells how to get to distant networks. The second kind tells how to get to local hosts. Associated with each table is the network interface to use to reach the destination, and certain other information.

When an IP packet arrives, its destination address is looked up in the routing table. If the packet is for a distant network, it is forwarded to the next router on the interface given in the table. If it is a local host (e.g., on the router's LAN), it is sent directly to the destination. If the network is not present, the packet is forwarded to a default router with more extensive tables. This algorithm means that each router only has to keep track of other networks and local hosts, not (network, host) pairs, greatly reducing the size of the routing table.

When subnetting is introduced, the routing tables are changed, adding entries of the form (this-network, subnet, 0) and (this-network, this-subnet, host). Thus a router on subnet *k* knows how to get to all the other subnets and also how to get to all the hosts on subnet *k*. It does not have to know the details about hosts on other subnets. In fact, all that needs to be changed is to have each router do a Boolean AND with the network's **subnet mask** (see Fig. 5-49) to get rid of the host number and look up the resulting address in its tables (after determining which network class it is). For example, a packet addressed to 130.50.15.6 and arriving at a router on subnet 5 is ANDed with the subnet mask of Fig. 5-49 to give the address 130.50.12.0. This address is looked up in the routing tables to find out how to get to hosts on subnet 3. The router on subnet 5 is thus spared the work of keeping track of the data link addresses of hosts other than those on subnet 5. Subnetting thus reduces router table space by creating a three-level hierarchy.

5.5.4. Internet Control Protocols

In addition to IP, which is used for data transfer, the Internet has several control protocols used in the network layer, including ICMP, ARP, RARP, and BOOTP. In this section we will look at each of these in turn.

The Internet Control Message Protocol

The operation of the Internet is monitored closely by the routers. When something unexpected occurs, the event is reported by the **ICMP** (**Internet Control Message Protocol**), which is also used to test the Internet. About a dozen types of ICMP messages are defined. The most important ones are listed in Fig. 5-50. Each ICMP message type is encapsulated in an IP packet.

The DESTINATION UNREACHABLE message is used when the subnet or a router cannot locate the destination, or a packet with the *DF* bit cannot be delivered because a "small-packet" network stands in the way.

The TIME EXCEEDED message is sent when a packet is dropped due to its counter reaching zero. This event is a symptom that packets are looping, that there is enormous congestion, or that the timer values are being set too low.

Message type	Description
Destination unreachable	Packet could not be delivered
Time exceeded	Time to live field hit 0
Parameter problem	Invalid header field
Source quench	Choke packet
Redirect	Teach a router about geography
Echo request	Ask a machine if it is alive
Echo reply	Yes, I am alive
Timestamp request	Same as Echo request, but with timestamp
Timestamp reply	Same as Echo reply, but with timestamp

Fig. 5-50. The principal ICMP message types.

The PARAMETER PROBLEM message indicates that an illegal value has been detected in a header field. This problem indicates a bug in the sending host's IP software, or possibly in the software of a router transited.

The SOURCE QUENCH message was formerly used to throttle hosts that were sending too many packets. When a host received this message, it was expected to slow down. It is rarely used any more because when congestion occurs, these packets tend to add more fuel to the fire. Congestion control in the Internet is now done largely in the transport layer and will be studied in detail in Chap. 6.

The REDIRECT message is used when a router notices that a packet seems to be routed wrong. It is used by the router to tell the sending host about the probable error.

The ECHO REQUEST and ECHO REPLY messages are used to see if a given destination is reachable and alive. Upon receiving the ECHO message, the destination is expected to send an ECHO REPLY message back. The TIMESTAMP REQUEST and TIMESTAMP REPLY messages are similar, except that the arrival time of the message and the departure time of the reply are recorded in the reply. This facility is used to measure network performance.

In addition to these messages, there are four others that deal with Internet addressing, to allow hosts to discover their network numbers and to handle the case of multiple LANs sharing a single IP address. ICMP is defined in RFC 792.

The Address Resolution Protocol

Although every machine on the Internet has one (or more) IP addresses, these cannot actually be used for sending packets because the data link layer hardware does not understand Internet addresses. Nowadays, most hosts are attached to a

LAN by an interface board that only understands LAN addresses. For example, every Ethernet board ever manufactured comes equipped with a 48-bit Ethernet address. Manufacturers of Ethernet boards request a block of addresses from a central authority to ensure that no two boards have the same address (to avoid conflicts should the two boards ever appear on the same LAN). The boards send and receive frames based on 48-bit Ethernet addresses. They know nothing at all about 32-bit IP addresses.

The question now arises: How do IP addresses get mapped onto data link layer addresses, such as Ethernet? To explain how this works, let us use the example of Fig. 5-51, in which a small university with several class C networks is illustrated. Here we have two Ethernets, one in the Computer Science department, with IP address 192.31.65.0 and one in Electrical Engineering, with IP address 192.31.63.0. These are connected by a campus FDDI ring with IP address 192.31.60.0. Each machine on an Ethernet has a unique Ethernet address, labeled *E1* through *E6*, and each machine on the FDDI ring has an FDDI address, labeled *F1* through *F3*.

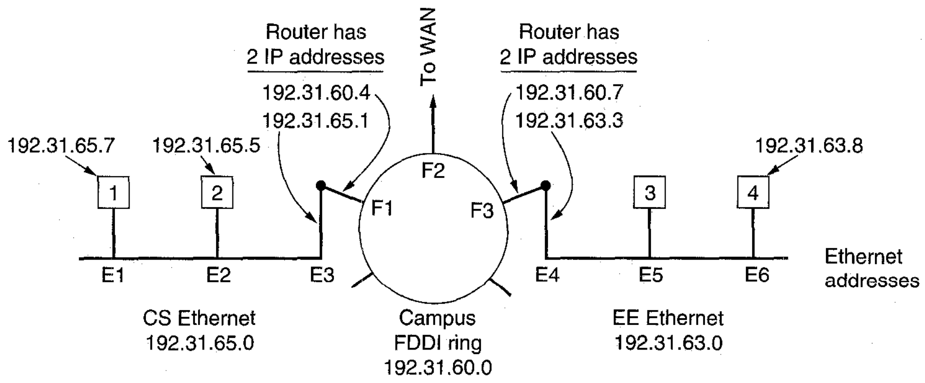


Fig. 5-51. Three interconnected class C networks: two Ethernets and an FDDI ring.

Let us start out by seeing how a user on host 1 sends a packet to a user on host 2. Let us assume the sender knows the name of the intended receiver, possibly something like *mary@eagle.cs.uni.edu*. The first step is to find the IP address for host 2, known as *eagle.cs.uni.edu*. This lookup is performed by the Domain Name System, which we will study in Chap. 7. For the moment, we will just assume that DNS returns the IP address for host 2 (192.31.65.5).

The upper layer software on host 1 now builds a packet with 192.31.65.5 in the *Destination address* field and gives it to the IP software to transmit. The IP software can look at the address and see that the destination is on its own network, but it needs a way to find the destination's Ethernet address. One solution is to have a configuration file somewhere in the system that maps IP addresses onto

Ethernet addresses. This solution is certainly possible, but for organizations with thousands of machines, keeping these files up to date is an error-prone, time-consuming job.

A better solution is for host 1 to output a broadcast packet onto the Ethernet asking: "Who owns IP address 192.31.65.5?" The broadcast will arrive at every machine on Ethernet 192.31.65.0, and each one will check its IP address. Host 2 alone will respond with its Ethernet address (*E2*). In this way host 1 learns that IP address 192.31.65.5 is on the host with Ethernet address *E2*. The protocol for asking this question and getting the reply is called **ARP** (**Address Resolution Protocol**). Almost every machine on the Internet runs it. It is defined in RFC 826.

The advantage of using ARP over configuration files is the simplicity. The system manager does not have to do much except assign each machine an IP address and decide about subnet masks. ARP does the rest.

At this point, the IP software on host 1 builds an Ethernet frame addressed to *E2*, puts the IP packet (addressed to 192.31.65.5) in the payload field, and dumps it onto the Ethernet. The Ethernet board of host 2 detects this frame, recognizes it as a frame for itself, scoops it up, and causes an interrupt. The Ethernet driver extracts the IP packet from the payload and passes it to the IP software, which sees that it is correctly addressed, and processes it.

Various optimizations are possible to make ARP more efficient. To start with, once a machine has run ARP, it caches the result in case it needs to contact the same machine shortly. Next time it will find the mapping in its own cache, thus eliminating the need for a second broadcast. In many cases host 2 will need to send back a reply, forcing it, too, to run ARP to determine the sender's Ethernet address. This ARP broadcast can be avoided by having host 1 include its IP to Ethernet mapping in the ARP packet. When ARP broadcast arrives at host 2, the pair (192.31.65.7, E1) is entered into host 2's ARP cache for future use. In fact, all machines on the Ethernet can enter this mapping into their ARP caches.

Yet another optimization is to have every machine broadcast its mapping when it boots. This broadcast is generally done in the form of an ARP looking for its own IP address. There should not be a response, but a side effect of the broadcast is to make any entry in everyone's ARP cache. If a response does arrive, two machines have been assigned the same IP address. The new one should inform the system manager and not boot.

To allow mappings to change, for example, when an Ethernet board breaks and is replaced with a new one (and thus a new Ethernet address), entries in the ARP cache should time out after a few minutes.

Now let us look at Fig. 5-51 again, only this time host 1 wants to send a packet to host 6 (192.31.63.8). Using ARP will fail because host 4 will not see the broadcast (routers do not forward Ethernet-level broadcasts). There are two solutions. First, the CS router could be configured to respond to ARP requests for network 192.31.63.0 (and possibly other local networks). In this case, host 1 will make an ARP cache entry of (192.31.63.8, E3) and happily send all traffic for host

4 to the local router. This solution is called **proxy ARP**. The second solution is to have host 1 immediately see that the destination is on a remote network and just send all such traffic to a default Ethernet address that handles all remote traffic, in this case *E3*. This solution does not require having the CS router know which remote networks it is serving.

Either way, what happens is that host 1 packs the IP packet into the payload field of an Ethernet frame addressed to *E3*. When the CS router gets the Ethernet frame, it removes the IP packet from the payload field and looks up the IP address in its routing tables. It discovers that packets for network 192.31.63.0 are supposed to go to router 192.31.60.7. If it does not already know the FDDI address of 192.31.60.7, it broadcasts an ARP packet onto the ring and learns that its ring address is *F3*. It then inserts the packet into the payload field of an FDDI frame addressed to *F3* and puts it on the ring.

At the EE router, the FDDI driver removes the packet from the payload field and gives it to the IP software, which sees that it needs to send the packet to 192.31.63.8. If this IP address is not in its ARP cache, it broadcasts an ARP request on the EE Ethernet and learns that the destination address is *E6* so it builds an Ethernet frame addressed to *E6*, puts the packet in the payload field, and sends it over the Ethernet. When the Ethernet frame arrives at host 4, the packet is extracted from the frame and passed to the IP software for processing.

Going from host 1 to a distant network over a WAN works essentially the same way, except that this time the CS router's tables tell it to use the WAN router whose FDDI address is *F2*.

The Reverse Address Resolution Protocol

ARP solves the problem of finding out which Ethernet address corresponds to a given IP address. Sometimes the reverse problem has to solved: Given an Ethernet address, what is the corresponding IP address? In particular, this problem occurs when booting a diskless workstation. Such a machine will normally get the binary image of its operating system from a remote file server. But how does it learn its IP address?

The solution is to use the **RARP** (**Reverse Address Resolution Protocol**) (defined in RFC 903). This protocol allows a newly-booted workstation to broadcast its Ethernet address and say: "My 48-bit Ethernet address is 14.04.05.18.01.25. Does anyone out there know my IP address?" The RARP server sees this request, looks up the Ethernet address in its configuration files, and sends back the corresponding IP address.

Using RARP is better than embedding an IP address in the memory image because it allows the same image to be used on all machines. If the IP address were buried inside the image, each workstation would need its own image.

A disadvantage of RARP is that it uses a destination address of all 1s (limited broadcasting) to reach the RARP server. However, such broadcasts are not forwarded by routers, so a RARP server is needed on each network. To get around this problem, an alternative bootstrap protocol called **BOOTP** has been invented (see RFCs 951, 1048, and 1084). Unlike RARP, it uses UDP messages, which are forwarded over routers. It also provides a diskless workstation with additional information, including the IP address of the file server holding the memory image, the IP address of the default router, and the subnet mask to use. BOOTP is described in RFC 951.

5.5.5. The Interior Gateway Routing Protocol: OSPF

As we mentioned earlier, the Internet is made up of a large number of autonomous systems. Each AS is operated by a different organization and can use its own routing algorithm inside. For example, the internal networks of companies *X*, *Y*, and *Z* would usually be seen as three ASes if all three were on the Internet. All three may use different routing algorithms internally. Nevertheless, having standards, even for internal routing, simplifies the implementation at the boundaries between ASes and allows reuse of code. In this section we will study routing within an AS. In the next one, we will look at routing between ASes. A routing algorithm within an AS is called an **interior gateway protocol**; an algorithm for routing between ASes is called an **exterior gateway protocol**.

The original Internet interior gateway protocol was a distance vector protocol (RIP) based on the Bellman-Ford algorithm. It worked well in small systems, but less well as ASes got larger. It also suffered from the count-to-infinity problem and generally slow convergence, so it was replaced in May 1979 by a link state protocol. In 1988, the Internet Engineering Task Force began work on a successor. That successor, called **OSPF** (**Open Shortest Path First**) became a standard in 1990. Many router vendors are now supporting it, and it will become the main interior gateway protocol in the near future. Below we will give a sketch of how OSPF works. For the complete story, see RFC 1247.

Given the long experience with other routing protocols, the group designing the new protocol had a long list of requirements that had to be met. First, the algorithm had to be published in the open literature, hence the "O" in OSPF. A proprietary solution owned by one company would not do. Second, the new protocol had to support a variety of distance metrics, including physical distance, delay, and so on. Third, it had to be a dynamic algorithm, one that adapted to changes in the topology automatically and quickly.

Fourth, and new for OSPF, it had to support routing based on type of service. The new protocol had to be able to route real-time traffic one way and other traffic a different way. The IP protocol has a *Type of Service* field, but no existing routing protocol used it.

Fifth, and related to the above, the new protocol had to do load balancing, splitting the load over multiple lines. Most previous protocols sent all packets over the best route. The second-best route was not used at all. In many cases, splitting the load over multiple lines gives better performance.

Sixth, support for hierarchical systems was needed. By 1988, the Internet had grown so large that no router could be expected to know the entire topology. The new routing protocol had to be designed so that no router would have to.

Seventh, some modicum of security was required to prevent fun-loving students from spoofing routers by sending them false routing information. Finally, provision was needed for dealing with routers that were connected to the Internet via a tunnel. Previous protocols did not handle this well.

OSPF supports three kinds of connections and networks:

1. Point-to-point lines between exactly two routers.
2. Multiaccess networks with broadcasting (e.g., most LANs).
3. Multiaccess networks without broadcasting (e.g., most packet-switched WANs).

A **multiaccess** network is one that can have multiple routers on it, each of which can directly communicate with all the others. All LANs and WANs have this property. Figure 5-52(a) shows an AS containing all three kinds of networks. Note that hosts do not generally play a role in OSPF.

OSPF works by abstracting the collection of actual networks, routers, and lines into a directed graph in which each arc is assigned a cost (distance, delay, etc.). It then computes the shortest path based on the weights on the arcs. A serial connection between two routers is represented by a pair of arcs, one in each direction. Their weights may be different. A multiaccess network is represented by a node for the network itself plus a node for each router. The arcs from the network node to the routers have weight 0 and are omitted from the graph.

Figure 5-52(b) shows the graph representation of the network of Fig. 5-52(a). What OSPF fundamentally does is represent the actual network as a graph like this and then compute the shortest path from every router to every other router.

Many of the ASes in the Internet are themselves large and nontrivial to manage. OSPF allows them to be divided up into numbered **areas**, where an area is a network or a set of contiguous networks. Areas do not overlap but need not be exhaustive, that is, some routers may belong to no area. An area is a generalization of a subnet. Outside an area, its topology and details are not visible.

Every AS has a **backbone** area, called area 0. All areas are connected to the backbone, possibly by tunnels, so it is possible to go from any area in the AS to any other area in the AS via the backbone. A tunnel is represented in the graph as an arc and has a cost. Each router that is connected to two or more areas is part of the backbone. As with other areas, the topology of the backbone is not visible outside the backbone.

Within an area, each router has the same link state database and runs the same shortest path algorithm. Its main job is to calculate the shortest path from itself to

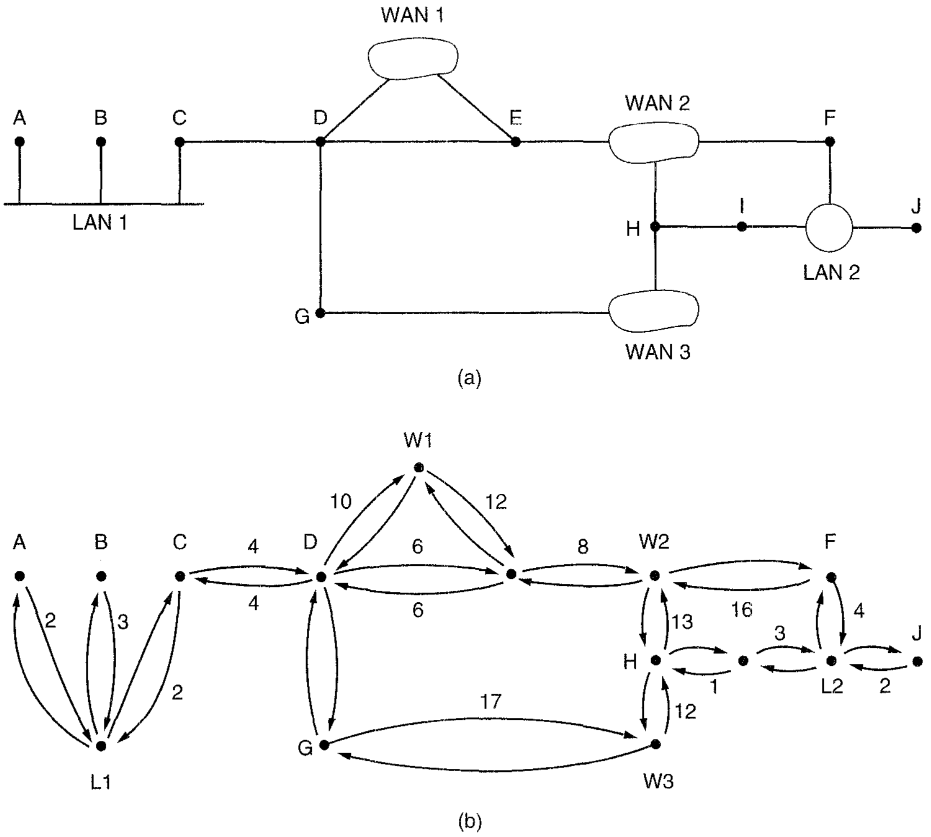


Fig. 5-52. (a) An autonomous system. (b) A graph representation of (a).

every other router in the area, including the router that is connected to the backbone, of which there must be at least one. A router that connects to two areas needs the databases for both areas and must run the shortest path algorithm for each one separately.

The way OSPF handles type of service routing is to have multiple graphs, one labeled with the costs when delay is the metric, one labeled with the costs when throughput is the metric, and one labeled with the costs when reliability is the metric. Although this triples the computation needed, it allows separate routes for optimizing delay, throughput, and reliability.

During normal operation, three kinds of routes may be needed: intra-area, interarea, and interAS. Intra-area routes are the easiest, since the source router already knows the shortest path to the destination router. Interarea routing always proceeds in three steps: go from the source to the backbone; go across the backbone to the destination area; go to the destination. This algorithm forces a star configuration on OSPF with the backbone being the hub and the other areas being spokes. Packets are routed from source to destination "as is." They are not encapsulated or tunneled, unless going to an area whose only connection to the backbone is a tunnel. Figure 5-53 shows part of the Internet with ASes and areas.

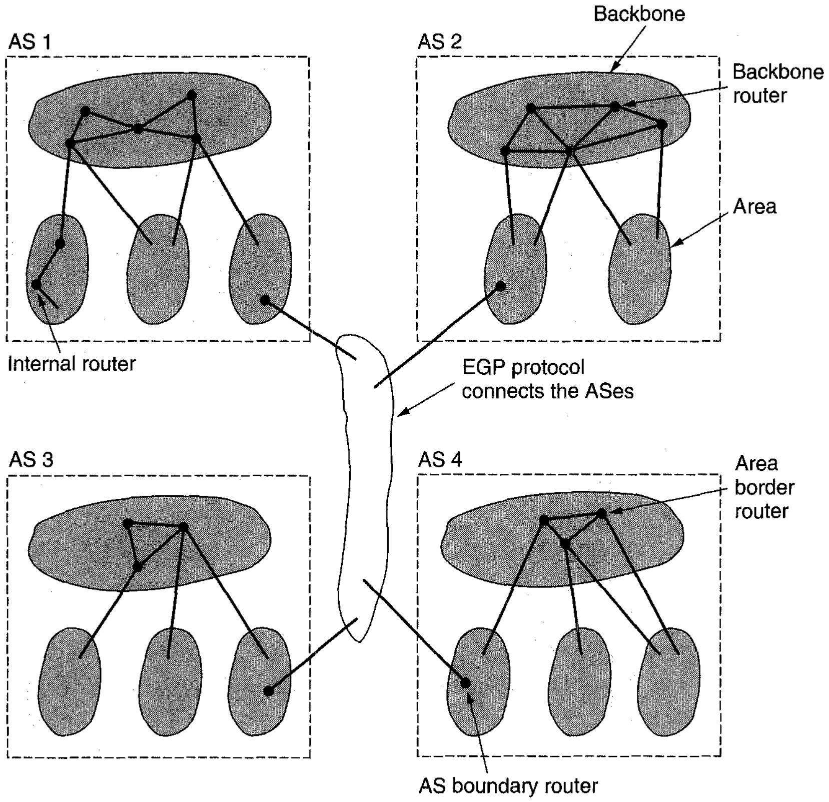


Fig. 5-53. The relation between ASes, backbones, and areas in OSPF.

OSPF distinguishes four classes of routers:

1. Internal routers are wholly within one area.
2. Area border routers connect two or more areas.
3. Backbone routers are on the backbone.
4. AS boundary routers talk to routers in other ASes.

These classes are allowed to overlap. For example, all the border routers are automatically part of the backbone. In addition, a router that is in the backbone but not part of any other area is also an internal router. Examples of all four classes of routers are illustrated in Fig. 5-53.

When a router boots, it sends HELLO messages on all of its point-to-point lines and multicasts them on LANs to the group consisting of all the other routers. On WANs, it needs some configuration information to know who to contact. From the responses, each router learns who its neighbors are.

OSPF works by exchanging information between **adjacent** routers, which is not the same as between neighboring routers. In particular, it is inefficient to have every router on a LAN talk to every other router on the LAN. To avoid this situation, one router is elected as the **designated router**. It is said to be adjacent to all the other routers, and exchanges information with them. Neighboring routers that are not adjacent do not exchange information with each other. A backup designated router is always kept up to date to ease the transition should the primary designated router crash.

During normal operation, each router periodically floods LINK STATE UPDATE messages to each of its adjacent routers. This message gives its state and provides the costs used in the topological database. The flooding messages are acknowledged, to make them reliable. Each message has a sequence number, so a router can see whether an incoming LINK STATE UPDATE is older or newer than what it currently has. Routers also send these messages when a line goes up or down or its cost changes.

DATABASE DESCRIPTION messages give the sequence numbers of all the link state entries currently held by the sender. By comparing its own values with those of the sender, the receiver can determine who has the most recent values. These messages are used when a line is brought up.

Either partner can request link state information from the other one using LINK STATE REQUEST messages. The net result of this algorithm is that each pair of adjacent routers checks to see who has the most recent data, and new information is spread throughout the area this way. All these messages are sent as raw IP packets. The five kinds of messages are summarized in Fig. 5-54.

Message type	Description
Hello	Used to discover who the neighbors are
Link state update	Provides the sender's costs to its neighbors
Link state ack	Acknowledges link state update
Database description	Announces which updates the sender has
Link state request	Requests information from the partner

Fig. 5-54. The five types of OSPF messages.

Finally, we can put all the pieces together. Using flooding, each router informs all the other routers in its area of its neighbors and costs. This information allows each router to construct the graph for its area(s) and compute the shortest path. The backbone area does this too. In addition, the backbone routers accept information from the area border routers in order to compute the best route from each backbone router to every other router. This information is propagated back to the area border routers, which advertise it within their areas. Using this information, a router about to send an interarea packet can select the best exit router to the backbone.

5.5.6. The Exterior Gateway Routing Protocol: BGP

Within a single AS, the recommended routing protocol on the Internet is OSPF (although it is certainly not the only one in use). Between ASes, a different protocol, **BGP** (**Border Gateway Protocol**), is used. A different protocol is needed between ASes because the goals of an interior gateway protocol and an exterior gateway protocol are not the same. All an interior gateway protocol has to do is move packets as efficiently as possible from the source to the destination. It does not have to worry about politics.

Exterior gateway protocol routers have to worry about politics a great deal. For example, a corporate AS might want the ability to send packets to any Internet site and receive packets from any Internet site. However, it might be unwilling to carry transit packets originating in a foreign AS and ending in a different foreign AS, even if its own AS was on the shortest path between the two foreign ASes ("That's their problem, not ours"). On the other hand, it might be willing to carry transit traffic for its neighbors, or even for specific other ASes that paid it for this service. Telephone companies, for example, might be happy to act as a carrier for their customers, but not for others. Exterior gateway protocols in general, and BGP in particular, have been designed to allow many kinds of routing policies to be enforced in the interAS traffic.

Typical policies involve political, security, or economic considerations. A few examples of routing constraints are

1. No transit traffic through certain ASes.
2. Never put Iraq on a route starting at the Pentagon.
3. Do not use the United States to get from British Columbia to Ontario.
4. Only transit Albania if there is no alternative to the destination.
5. Traffic starting or ending at IBM® should not transit Microsoft®.

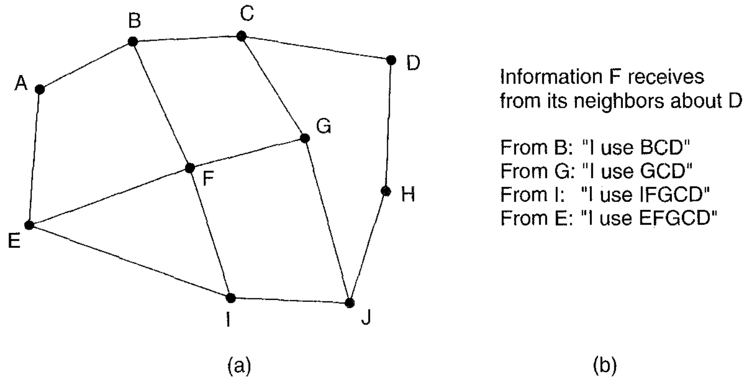
Policies are manually configured into each BGP router. They are not part of the protocol itself.

From the point of view of a BGP router, the world consists of other BGP routers and the lines connecting them. Two BGP routers are considered connected if they share a common network. Given BGP's special interest in transit traffic, networks are grouped into one of three categories. The first category is the **stub networks**, which have only one connection to the BGP graph. These cannot be used for transit traffic because there is no one on the other side. Then come the **multiconnected networks**. These could be used for transit traffic, except that they refuse. Finally, there are the **transit networks**, such as backbones, which are willing to handle third-party packets, possibly with some restrictions.

Pairs of BGP routers communicate with each other by establishing TCP connections. Operating this way provides reliable communication and hides all the details of the network being passed through.

BGP is fundamentally a distance vector protocol, but quite different from most others such as RIP. Instead of maintaining just the cost to each destination, each BGP router keeps track of the exact path used. Similarly, instead of periodically giving each neighbor its estimated cost to each possible destination, each BGP router tells its neighbors the exact path it is using.

As an example, consider the BGP routers shown in Fig. 5-55(a). In particular, consider *F*'s routing table. Suppose that it uses the path *FGCD* to get to *D*. When the neighbors give it routing information, they provide their complete paths, as shown in Fig. 5-55(b) (for simplicity, only destination *D* is shown here).



Information F receives
from its neighbors about D

From B: "I use BCD"
From G: "I use GCD"
From I: "I use IFGCD"
From E: "I use EFGCD"

Fig. 5-55. (a) A set of BGP routers. (b) Information sent to *F*.

After all the paths come in from the neighbors, *F* examines them to see which is the best. It quickly discards the paths from *I* and *E*, since these paths pass through *F* itself. The choice is then between using *B* and *G*. Every BGP router contains a module that examines routes to a given destination and scores them, returning a number for the "distance" to that destination for each route. Any route violating a policy constraint automatically gets a score of infinity. The router then adopts the route with the shortest distance. The scoring function is not part of the BGP protocol and can be any function the system managers want.

BGP easily solves the count-to-infinity problem that plagues other distance vector routing algorithms. For example, suppose *G* crashes or the line *FG* goes down. *F* then receives routes from its three remaining neighbors. These routes are *BCD*, *IFGCD*, and *EFGCD*. It can immediately see that the two latter routes are pointless, since they pass through *F* itself, so it chooses *FBCD* as its new route. Other distance vector algorithms often make the wrong choice because they cannot tell which of their neighbors have independent routes to the destination, and which do not. The current definition of BGP is in RFC 1654. Additional useful information can be found in RFC 1268.

5.5.7. Internet Multicasting

Normal IP communication is between one sender and one receiver. However, for some applications it is useful for a process to be able to send to a large number of receivers simultaneously. Examples are updating replicated, distributed databases, transmitting stock quotes to multiple brokers, and handling digital conference (i.e., multiparty) telephone calls.

IP supports multicasting, using class D addresses. Each class D address identifies a group of hosts. Twenty-eight bits are available for identifying groups, so over 250 million groups can exist at the same time. When a process sends a packet to a class D address, a best-efforts attempt is made to deliver it to all the members of the group addressed, but no guarantees are given. Some members may not get the packet.

Two kinds of group addresses are supported: permanent addresses and temporary ones. A permanent group is always there and does not have to be set up. Each permanent group has a permanent group address. Some examples of permanent group addresses are

224.0.0.1	All systems on a LAN
224.0.0.2	All routers on a LAN
224.0.0.5	All OSPF routers on a LAN
224.0.0.6	All designated OSPF routers on a LAN

Temporary groups must be created before they can be used. A process can ask its host to join a specific group. It can also ask its host to leave the group. When the last process on a host leaves a group, that group is no longer present on the host. Each host keeps track of which groups its processes currently belong to.

Multicasting is implemented by special multicast routers, which may or may not be colocated with the standard routers. About once a minute, each multicast router sends a hardware (i.e., data link layer) multicast to the hosts on its LAN (address 224.0.0.1) asking them to report back on the groups their processes currently belong to. Each host sends back responses for all the class D addresses it is interested in.

These query and response packets use a protocol called **IGMP (Internet Group Management Protocol)**, which is vaguely analogous to ICMP. It has only two kinds of packets: query and response, each with a simple fixed format containing some control information in the first word of the payload field and a class D address in the second word. It is described in RFC 1112.

Multicast routing is done using spanning trees. Each multicast router exchanges information with its neighbors using a modified distance vector protocol in order for each one to construct a spanning tree per group covering all group members. Various optimizations are used to prune the tree to eliminate routers and networks not interested in particular groups. The protocol makes heavy use of tunneling to avoid bothering nodes not in a spanning tree.

5.5.8. Mobile IP

Many users of the Internet have portable computers and want to stay connected to the Internet when they visit a distant Internet site and even on the road in between. Unfortunately, the IP addressing system makes working far from home easier said than done. In this section we will examine the problem and the solution. A more detailed description is given in (Johnson, 1995).

The real villain is the addressing scheme itself. Every IP address contains three fields: the class, the network number, and the host number. For example, consider the machine with IP address 160.80.40.20. The 160.80 gives the class (B) and network number (8272); the 40.20 is the host number (10260). Routers all over the world have routing tables telling which line to use to get to network 160.80. Whenever a packet comes in with a destination IP address of the form 160.80.xxx.yyy, it goes out on that line.

If all of a sudden, the machine with that address is carted off to some distant site, the packets for it will continue to be routed to its home LAN (or router). The owner will no longer get email, and so on. Giving the machine a new IP address corresponding to its new location is unattractive because large numbers of people, programs, and databases would have to be informed of the change.

Another approach is to have the routers use complete IP addresses for routing, instead of just the class and network. However, this strategy would require each router to have millions of table entries, at astronomical cost to the Internet.

When people began demanding the ability to have mobile hosts, the IETF set up a Working Group to find a solution. The Working Group quickly formulated a number of goals considered desirable in any solution. The major ones were

1. Each mobile host must be able to use its home IP address anywhere.
2. Software changes to the fixed hosts were not permitted.
3. Changes to the router software and tables were not permitted.
4. Most packets for mobile hosts should not make detours on the way.
5. No overhead should be incurred when a mobile host is at home.

The solution chosen was the one described in Sec. 5.2.8. To review it briefly, every site that wants to allow its users to roam has to create a home agent. Every site that wants to allow visitors has to create a foreign agent. When a mobile host shows up at a foreign site, it contacts the foreign host there and registers. The foreign host then contacts the user's home agent and gives it a **care-of address**, normally the foreign agent's own IP address.

When a packet arrives at thc user's home LAN, it comes in at some router attached to the LAN. The router then tries to locate the host in the usual way, by broadcasting an ARP packet asking, for example: "What is the Ethernet address of 160.80.40.20?" The home agent responds to this query by giving its own Ethernet address. The router then sends packets for 160.80.40.20 to the home agent. It, in turn, tunnels them to the care-of address by encapsulating them in the payload field of an IP packet addressed to the foreign agent. The foreign agent then decapsulates and delivers them to the data link address of the mobile host. In addition, the home agent gives the care-of address to the sender, so future packets can be tunnelled directly to the foreign agent. This solution meets all the requirements stated above.

One small detail is probably worth mentioning. At the time the mobile host moves, the router probably has its (soon-to-be-invalid) Ethernet address cached. To replace that Ethernet address with the home agent's, a trick called **gratuitious ARP** is used. This is a special, unsolicited message to the router that causes it to replace a specific cache entry, in this case, that of the mobile host about to leave. When the mobile host returns later, the same trick is used to update the router's cache again.

Nothing in the design prevents a mobile host from being its own foreign agent, but that approach only works if the mobile host (in its capacity as foreign agent) is logically connected to the Internet at its current site. Also, it must be able to acquire a (temporary) care-of IP address to use. That IP address must belong to the LAN to which it is currently attached.

The IETF solution for mobile hosts solves a number of other problems not mentioned so far. For example, how are agents located? The solution is for each agent to periodically broadcast its address and the type of services it is willing to provide (e.g., home, foreign, or both). When a mobile host arrives somewhere, it can just listen for these broadcasts, called **advertisements**. Alternatively, it can broadcast a packet announcing its arrival and hope that the local foreign agent responds to it.

Another problem that had to be solved is what to do about impolite mobile hosts that leave without saying goodbye. The solution is to make registration valid only for a fixed time interval. If it is not refreshed periodically, it times out, so the foreign host can clear its tables.

Yet another issue is security. When a home agent gets a message asking it to please forward all of Nora's packets to some IP address, it had better not comply unless it is convinced that Nora is the source of this request, and not somebody trying to impersonate her. Cryptographic authentication protocols are used for this purpose. We will study such protocols in Chap. 7.

A final point addressed by the Working Group relates to levels of mobility. Imagine an airplane with an on-board Ethernet used by the navigation and avionics computers. On this Ethernet is a standard router that talks to the wired Internet on the ground over a radio link. One fine day, some clever marketing executive gets the idea to install Ethernet connectors in all the arm rests so passengers with mobile computers can also plug in.

Now we have two levels of mobility: the aircraft's own computers, which are stationary with respect to the Ethernet, and the passengers' computers, which are mobile with respect to it. In addition, the on-board router is mobile with respect to routers on the ground. Being mobile with respect to a system that is itself mobile can be handled using recursive tunneling.

5.5.9. CIDR—Classless InterDomain Routing

IP has been in heavy use for over a decade. It has worked extremely well, as demonstrated by the exponential growth of the Internet. Unfortunately, IP is rapidly becoming a victim of its own popularity: it is running out of addresses. This looming disaster has sparked a great deal of discussion and controversy within the Internet community about what to do about it. In this section we will describe both the problem and several proposed solutions. A more complete description is given in (Huitema, 1996).

Back in 1987, a few visionaries predicted that some day the Internet might grow to 100,000 networks. Most experts pooh-poohed this as being decades in the future, if ever. The 100,000th network was connected in 1996. The problem, simply stated, is that the Internet is rapidly running out of IP addresses. In principle, over 2 billion addresses exist, but the practice of organizing the address space by classes (see Fig. 5-47), wastes millions of them. In particular, the real villain is the class B network. For most organizations, a class A network, with 16 million addresses is too big, and a class C network, with 256 addresses is too small. A class B network, with 65,536, is just right. In Internet folklore, this situation is known as the **three bears problem** (as in *Goldilocks and the Three Bears*).

In reality, a class B address is far too large for most organizations. Studies have shown that more than half of all class B networks have fewer than 50 hosts. A class C network would have done the job, but no doubt every organization that asked for a class B address thought that one day it would outgrow the 8-bit host field. In retrospect, it might have been better to have had class C networks use 10 bits instead of eight for the host number, allowing 1022 hosts per network. Had this been the case, most organizations would have probably settled for a class C network, and there would have been half a million of them (versus only 16,384 class B networks).

However, then another problem would have emerged more quickly: the routing table explosion. From the point of view of the routers, the IP address space is a two-level hierarchy, with network numbers and host numbers. Routers do not have to know about all the hosts, but they do have to know about all the networks. If half a million class C networks were in use, every router in the entire Internet would need a table with half a million entries, one per network, telling which line to use to get to that network, as well as other information.

The actual physical storage of half a million entry tables is probably doable, although expensive for critical routers that keep the tables in static RAM on I/O boards. A more serious problem is that the complexity of various algorithms relating to management of the tables grows faster than linear. Worse yet, much of the existing router software and firmware was designed at a time when the Internet had 1000 connected networks and 10,000 networks seemed decades away. Design choices made then often are far from optimal now.

In addition, various routing algorithms require each router to transmit its tables periodically. The larger the tables, the more likely some parts will get lost underway, leading to incomplete data at the other end and possibly routing instabilities.

The routing table problem could have been solved by going to a deeper hierarchy. For example, having each IP address contain a country, state, city, network, and host field might work. Then each router would only need to know how to get to each country, the states or provinces in its own country, the cities in its state or province, and the networks in its city. Unfortunately, this solution would require considerably more than 32 bits for IP addresses and would use addresses inefficiently (Liechtenstein would have as many bits as the United States).

In short, most solutions solve one problem but create a new one. One solution that is now being implemented and which will give the Internet a bit of extra breathing room is **CIDR (Classless InterDomain Routing)**. The basic idea behind CIDR, which is described in RFC 1519, is to allocate the remaining class C networks, of which there are almost two million, in variable-sized blocks. If a site needs, say, 2000 addresses, it is given a block of 2048 addresses (eight contiguous class C networks), and not a full class B address. Similarly, a site needing 8000 addresses gets 8192 addresses (32 contiguous class C networks).

In addition to using blocks of contiguous class C networks as units, the allocation rules for the class C addresses were also changed in RFC 1519. The world was partitioned into four zones, and each one given a portion of the class C address space. The allocation was as follows:

Addresses 194.0.0.0 to 195.255.255.255 are for Europe
Addresses 198.0.0.0 to 199.255.255.255 are for North America
Addresses 200.0.0.0 to 201.255.255.255 are for Central and South America
Addresses 202.0.0.0 to 203.255.255.255 are for Asia and the Pacific

In this way, each region was given about 32 million addresses to allocate, with another 320 million class C addresses from 204.0.0.0 through 223.255.255.255

held in reserve for the future. The advantage of this allocation is that now any router outside of Europe that gets a packet addressed to 194.xx.yy.zz or 195.xx.yy.zz can just send it to its standard European gateway. In effect 32 million addresses have now been compressed into one routing table entry. Similarly for the other regions.

Of course, once a 194.xx.yy.zz packet gets to Europe, more detailed routing tables are needed. One possibility is to have 131,072 entries for networks 194.0.0.xx through 195.255.255.xx, but this is precisely this routing table explosion that we are trying to avoid. Instead, each routing table entry is extended by giving it a 32-bit mask. When a packet comes in, its destination address is first extracted. Then (conceptually) the routing table is scanned entry by entry, masking the destination address and comparing it to the table entry looking for a match.

To make this comparison process clearer, let us consider an example. Suppose that Cambridge University needs 2048 addresses and is assigned the addresses 194.24.0.0 through 194.24.7.255, along with mask 255.255.248.0. Next, Oxford University asks for 4096 addresses. Since a block of 4096 addresses must lie on a 4096-byte boundary, they cannot be given addresses starting at 194.8.0.0. Instead they get 194.24.16.0 through 194.24.31.255 along with mask 255.255.240.0. Now the University of Edinburgh asks for 1024 addresses and is assigned addresses 194.24.8.0 through 194.24.11.255 and mask 255.255.252.0.

The routing tables all over Europe are now updated with three entries, each one containing a base address and a mask. These entries (in binary) are:

Address	Mask
11000010 00011000 00000000 00000000	11111111 11111111 11111000 00000000
11000010 00011000 00010000 00000000	11111111 11111111 11110000 00000000
11000010 00011000 00001000 00000000	11111111 11111111 11111100 00000000

Now consider what happens when a packet comes in addressed to 194.24.17.4, which in binary is

11000010 00011000 00010001 00000100

First it is Boolean ANDed with the Cambridge mask to get

11000010 00011000 00010000 00000000

This value does not match the Cambridge base address, so the original address is next ANDed with the Oxford mask to get

11000010 00011000 00010000 00000000

This value does match the Oxford mask, so the packet is sent to the Oxford router. In practice, the router entries are not tried sequentially; indexing tricks are used to speed up the search. Also, it is possible for two entries to match, in which case the one whose mask has the most 1 bits wins. Finally, the same idea can be applied to all addresses, not just the new class C addresses, so with CIDR, the old class A, B, and C networks are no longer used for routing. This is why CIDR is called classless routing. CIDR is described in more detail in (Ford et al., 1993; and Huitema, 1995).

5.5.10. IPv6

While CIDR may buy a few more years' time, everyone realizes that the days of IP in its current form (IPv4) are numbered. In addition to these technical problems, there is another issue looming in the background. Up until recently, the Internet has been used largely by universities, high-tech industry, and the government (especially the Dept. of Defense). With the explosion of interest in the Internet starting in the mid 1990s, it is likely that in the next millenium, it will be used by a much larger group of people, especially people with different requirements. For one thing, millions of people with wireless portables may use it to keep in contact with their home bases. For another, with the impending convergence of the computer, communication, and entertainment industries, it may not be long before every television set in the world is an Internet node, producing a billion machines being used for video on demand. Under these circumstances, it became apparent that IP had to evolve and become more flexible.

Seeing these problems on the horizon, in 1990, IETF started work on a new version of IP, one which would never run out of addresses, would solve a variety of other problems, and be more flexible and efficient as well. Its major goals were to

1. Support billions of hosts, even with inefficient address space allocation.
2. Reduce the size of the routing tables.
3. Simplify the protocol, to allow routers to process packets faster.
4. Provide better security (authentication and privacy) than current IP.
5. Pay more attention to type of service, particularly for real-time data.
6. Aid multicasting by allowing scopes to be specified.
7. Make it possible for a host to roam without changing its address.
8. Allow the protocol to evolve in the future.
9. Permit the old and new protocols to coexist for years.

To find a protocol that met all these requirements, IETF issued a call for proposals and discussion in RFC 1550. Twenty-one responses were received, not all of them full proposals. By December 1992, seven serious proposals were on the table. They ranged from making minor patches to IP, to throwing it out altogether and replacing with a completely different protocol.

One proposal was to run TCP over CLNP, which, with its 160-bit addresses would have provided enough address space forever and would have unified two major network layer protocols. However, many people felt that this would have been an admission that something in the OSI world was actually done right, a statement considered Politically Incorrect in Internet circles. CLNP was patterned closely on IP, so the two are not really that different. In fact, the protocol ultimately chosen differs from IP far more than CLNP does. Another strike against CLNP was its poor support for service types, something required to transmit multimedia efficiently.

Three of the better proposals were published in *IEEE Network* (Deering, 1993; Francis, 1993; and Katz and Ford, 1993). After much discussion, revision, and jockeying for position, a modified combined version of the Deering and Francis proposals, by now called **SIPP** (**Simple Internet Protocol Plus**) was selected and given the designation **IPv6** (IPv5 was already in use for an experimental real-time stream protocol).

IPv6 meets the goals fairly well. It maintains the good features of IP, discards or deemphasizes the bad ones, and adds new ones where needed. In general, IPv6 is not compatible with IPv4, but it is compatible with all the other Internet protocols, including TCP, UDP, ICMP, IGMP, OSPF, BGP, and DNS, sometimes with small modifications being required (mostly to deal with longer addresses). The main features of IPv6 are discussed below. More information about it can be found in RFC 1883 through RFC 1887.

First and foremost, IPv6 has longer addresses than IPv4. They are 16 bytes long, which solves the problem that IPv6 was set out to solve: provide an effectively unlimited supply of Internet addresses. We will have more to say about addresses shortly.

The second major improvement of IPv6 is the simplification of the header. It contains only 7 fields (versus 13 in IPv4). This change allows routers to process packets faster and thus improve throughout. We will discuss the header shortly, too.

The third major improvement was better support for options. This change was essential with the new header because fields that previously were required are now optional. In addition, the way options are represented is different, making it simple for routers to skip over options not intended for them. This feature speeds up packet processing time.

A fourth area in which IPv6 represents a big advance is in security. IETF had its fill of newspaper stories about precocious 12-year-olds using their personal computers to break into banks and military bases all over the Internet. There was a strong feeling that something had to be done to improve security. Authentication and privacy are key features of the new IP.

Finally, more attention has been paid to type of service than in the past. IPv4 actually has an 8-bit field devoted to this matter, but with the expected growth in multimedia traffic in the future, much more is needed.

The Main IPv6 Header

The IPv6 header is shown in Fig. 5-56. The *Version* field is always 6 for IPv6 (and 4 for IPv4). During the transition period from IPv4, which will probably take a decade, routers will be able to examine this field to tell what kind of packet they have. As an aside, making this test wastes a few instructions in the critical path, so many implementations are likely to try to avoid it by using some field in the data link header to distinguish IPv4 packets from IPv6 packets. In this way, packets can be passed to the correct network layer handler directly. However, having the data link layer be aware of network packet types completely violates the design principle that each layer should not be aware of the meaning of the bits given to it from the layer above. The discussions between the "Do it right" and "Make it fast" camps will no doubt be lengthy and vigorous.

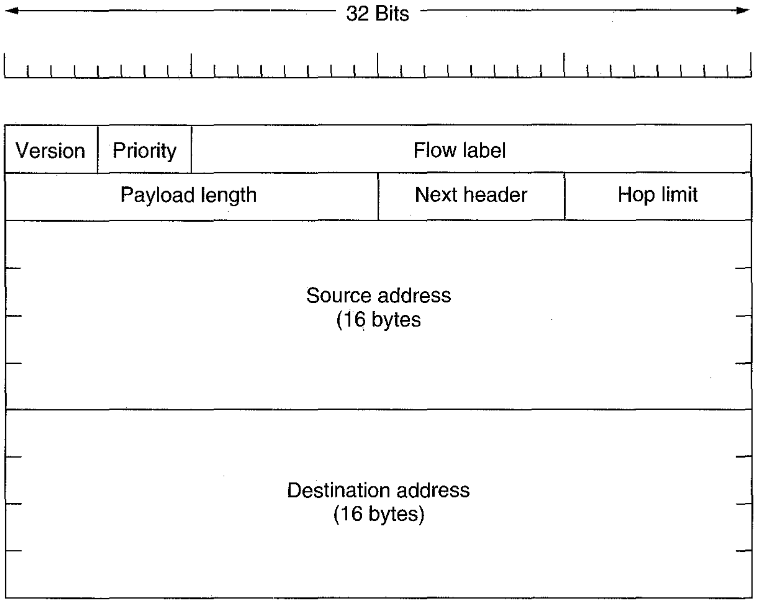


Fig. 5-56. The IPv6 fixed header (required).

The *Priority* field is used to distinguish between packets whose sources can be flow controlled and those that cannot. Values 0 through 7 are for transmissions that are capable of slowing down in the event of congestion. Values 8 through 15 are for real-time traffic whose sending rate is constant, even if all the packets are being lost. Audio and video fall into the latter category. This distinction allows routers to deal with packets better in the event of congestion. Within each group, lower-numbered packets are less important than higher-numbered ones. The IPv6 standard suggests, for example, to use 1 for news, 4 for FTP, and 6 for Telnet connections, since delaying a news packet for a few seconds is not noticeable, but delaying a Telnet packet certainly is.

The *Flow label* field is still experimental but will be used to allow a source and destination to set up a pseudoconnection with particular properties and requirements. For example, a stream of packets from one process on a certain source host to a certain process on a certain destination host might have stringent delay requirements and thus need reserved bandwidth. The flow can be set up in advance and given an identifier. When a packet with a nonzero *Flow label* shows up, all the routers can look it up in internal tables to see what kind of special treatment it requires. In effect, flows are an attempt to have it both ways: the flexibility of a datagram subnet and the guarantees of a virtual circuit subnet.

Each flow is designated by the source address, destination address, and flow number, so many flows may be active at the same time between a given pair of IP addresses. Also, in this way, even if two flows coming from different hosts but with the same flow number pass through the same router, the router will be able to tell them apart using the source and destination addresses. It is expected that flow numbers will be chosen randomly, rather than assigned sequentially starting at 1, to make it easy for routers to hash them.

The *Payload length* field tells how many bytes follow the 40-byte header of Fig. 5-56. The name was changed from the IPv4 *Total length* field because the meaning was changed slightly: the 40 header bytes are no longer counted as part of the length as they used to be.

The *Next header* field lets the cat out of the bag. The reason the header could be simplified is that there can be additional (optional) extension headers. This field tells which of the (currently) six extension headers, if any, follows this one. If this header is the last IP header, the *Next header* field tells which transport protocol handler (e.g., TCP, UDP) to pass the packet to.

The *Hop limit* field is used to keep packets from living forever. It is, in practice, the same as the *Time to live* field in IPv4, namely, a field that is decremented on each hop. In theory, in IPv4 it was a time in seconds, but no router used it that way, so the name was changed to reflect the way it is actually used.

Next come the *Source address* and *Destination address* fields. Deering's original proposal, SIP, used 8-byte addresses, but during the review process many people felt that with 8-byte addresses IPv6 would run out of addresses within a few decades, whereas with 16-byte addresses it would never run out. Other people argued that 16 bytes was overkill, whereas still others favored using 20-byte addresses to be compatible with the OSI datagram protocol. Another faction wanted variable-sized addresses. After much discussion, it was decided that fixed-length 16-byte addresses were the best compromise.

The IPv6 address space is divided up as shown in Fig. 5-57. Addresses beginning with 80 zeros are reserved for IPv4 addresses. Two variants are supported, distinguished by the next 16 bits. These variants relate to how IPv6 packets will be tunneled over the existing IPv4 infrastructure.

Prefix (binary)	Usage	Fraction
0000 0000	Reserved (including IPv4)	1/256
0000 0001	Unassigned	1/256
0000 001	OSI NSAP addresses	1/128
0000 010	Novell NetWare IPX addresses	1/128
0000 011	Unassigned	1/128
0000 1	Unassigned	1/32
0001	Unassigned	1/16
001	Unassigned	1/8
010	Provider-based addresses	1/8
011	Unassigned	1/8
100	Geographic-based addresses	1/8
101	Unassigned	1/8
110	Unassigned	1/8
1110	Unassigned	1/16
1111 0	Unassigned	1/32
1111 10	Unassigned	1/64
1111 110	Unassigned	1/128
1111 1110 0	Unassigned	1/512
1111 1110 10	Link local use addresses	1/1024
1111 1110 11	Site local use addresses	1/1024
1111 1111	Multicast	1/256

Fig. 5-57. IPv6 addresses

The use of separate prefixes for provider-based and geographic-based addresses is a compromise between two different visions of the future of the Internet. Provider-based addresses make sense if you think that in the future there will be some number of companies providing Internet service to customers, analogous to AT&T, MCI, Sprint, British Telecom, and so on providing telephone service now. Each of these companies will be given some fraction of the address space. The first 5 bits following the 010 prefix are used to indicate which registry to look the provider up in. Currently three registries are operating, for North America, Europe, and Asia. Up to 29 new registries can be added later.

Each registry is free to divide up the remaining 15 bytes as it sees fit. It is expected that many of them will use a 3-byte provider number, giving about 16 million providers, in order to allow large companies to act as their own provider. Another possibility is to use 1 byte to indicate national providers and let them do further allocation. In this manner, additional levels of hierarchy can be introduced as needed.

The geographic model is the same as the current Internet, in which providers do not play a large role. In this way, IPv6 can handle both kinds of addresses.

The link and site local addresses have only a local significance. They can be reused at each organization without conflict. They cannot be propagated outside organizational boundaries, making them well suited to organizations that currently use firewalls to wall themselves off from the rest of the Internet.

Multicast addresses have a 4-bit flag field and a 4-bit scope field following the prefix, then a 112-bit group identifier. One of the flag bits distinguishes permanent from transient groups. The scope field allows a multicast to be limited to the current link, site, organization, or planet. These four scopes are spread out over the 16 values to allow new scopes to be added later. For example, the planetary scope is 14, so code 15 is available to allow future expansion of the Internet to other planets, solar systems, and galaxies.

In addition to supporting the standard unicast (point-to-point) and multicast addresses, IPv6 also supports a new kind of addressing: anycast. **Anycasting** is like multicasting in that the destination is a group of addresses, but instead of trying to deliver the packet to all of them, it tries to deliver it to just one, usually the nearest one. For example, when contacting a group of cooperating file servers, a client can use anycast to reach the nearest one, without having to know which one that is. Anycasting uses regular unicast addresses. It is up to the routing system to choose the lucky host that gets the packet.

A new notation has been devised for writing 16-byte addresses. They are written as eight groups of four hexadecimal digits with colons between the groups, like this:

8000:0000:0000:0000:0123:4567:89AB:CDEF

Since many addresses will have many zeros inside them, three optimizations have been authorized. First, leading zeros within a group can be omitted, so 0123 can be written as 123. Second, one or more groups of 16 zeros can be replaced by a pair of colons. Thus the above address now becomes

8000::123:4567:89AB:CDEF

Finally, IPv4 addresses can be written as a pair of colons and an old dotted decimal number, for example

::192.31.20.46

Perhaps it is unnecessary to be so explicit about it, but there are a lot of 16-byte addresses. Specifically, there are 2^{128} of them, which is approximately

3×10^{38}. If the entire earth, land and water, were covered with computers, IPv6 would allow 7×10^{23} IP addresses per square meter. Students of chemistry will notice that this number is larger than Avogadro's number. While it was not the intention to give every molecule on the surface of the earth its own IP address, we are not that far off.

In practice, the address space will not be used efficiently, just as the telephone number address space is not (the area code for Manhattan, 212, is nearly full, but that for Wyoming, 307, is nearly empty). In RFC 1715, Huitema calculated that using the allocation of telephone numbers as a guide, even in the most pessimistic scenario, there will still be well over 1000 IP addresses per square meter of the earth's surface (land and water). In any likely scenario, there will be trillions of them per square meter. In short, it seems unlikely that we will run out in the foreseeable future. It is also worth noting that only 28 percent of the address space has been allocated so far. The other 72 percent is available for future purposes not yet thought of.

It is instructive to compare the IPv4 header (Fig. 5-45) with the IPv6 header (Fig. 5-56) to see what has been left out in IPv6. The *IHL* field is gone because the IPv6 header has a fixed length. The *Protocol* field was taken out because the *Next header* field tells what follows the last IP header (e.g., a UDP or TCP segment).

All the fields relating to fragmentation were removed because IPv6 takes a different approach to fragmentation. To start with, all IPv6 conformant hosts and routers must support packets of 576 bytes. This rule makes fragmentation less likely to occur in the first place. In addition, when a host sends an IPv6 packet that is too large, instead of fragmenting it, the router that is unable to forward it sends back an error message. This message tells the host to break up all future packets to that destination. Having the host send packets that are the right size in the first place is ultimately much more efficient than having the routers fragment them on the fly.

Finally, the *Checksum* field is gone because calculating it greatly reduces performance. With the reliable networks now used, combined with the fact that the data link layer and transport layers normally have their own checksums, the value of yet another checksum was not worth the performance price it extracted. Removing all these features has resulted in a lean and mean network layer protocol. Thus the goal of IPv6—a fast, yet flexible, protocol with plenty of address space—has been met by this design.

Extension Headers

Nevertheless, some of the missing fields are occasionally still needed so IPv6 has introduced the concept of an (optional) **extension header**. These headers can be supplied to provide extra information, but encoded in an efficient way. Six kinds of extension headers are defined at present, as listed in Fig. 5-58. Each one is optional, but if more than one is present, they must appear directly after the fixed header, and preferably in the order listed.

Extension header	Description
Hop-by-hop options	Miscellaneous information for routers
Routing	Full or partial route to follow
Fragmentation	Management of datagram fragments
Authentication	Verification of the sender's identity
Encrypted security payload	Information about the encrypted contents
Destination options	Additional information for the destination

Fig. 5-58. IPv6 extension headers.

Some of the headers have a fixed format; others contain a variable number of variable-length fields. For these, each item is encoded as a (Type, Length, Value) tuple. The *Type* is a 1-byte field telling which option this is. The *Type* values have been chosen so that the first 2 bits tell routers that do not know how to process the option what to do. The choices are: skip the option, discard the packet, discard the packet and send back an ICMP packet, and the same as the previous one, except do not send ICMP packets for multicast addresses (to prevent one bad multicast packet from generating millions of ICMP reports).

The *Length* is also a 1-byte field. It tells how long the value is (0 to 255 bytes). The *Value* is any information required, up to 255 bytes.

The hop-by-hop header is used for information that all routers along the path must examine. So far, one option has been defined: support of datagrams exceeding 64K. The format of this header is shown in Fig. 5-59.

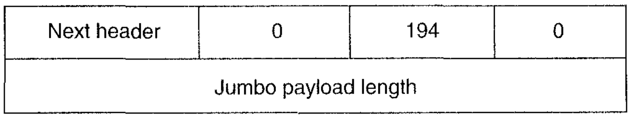


Fig. 5-59. The hop-by-hop extension header for large datagrams (jumbograms).

As with all extension headers, this one starts out with a byte telling what kind of header comes next. This byte is followed by one telling how long the hop-by-hop header is in bytes, excluding the first 8 bytes, which are mandatory. The next 2 bytes indicate that this option defines the datagram size (code 194) as a 4-byte number. The last 4 bytes give the size of the datagram. Sizes less than 65,536 are not permitted and will result in the first router discarding the packet and sending back an ICMP error message. Datagrams using this header extension are called

jumbograms. The use of jumbograms is important for supercomputer applications that must transfer gigabytes of data efficiently across the Internet.

The routing header lists one or more routers that must be visited on the way to the destination. Both strict routing (the full path is supplied) and loose routing (only selected routers are supplied) are available, but they are combined. The format of the routing header is shown in Fig. 5-60.

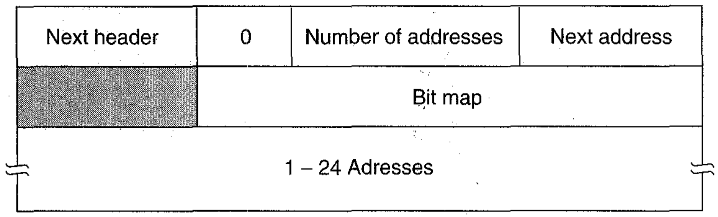


Fig. 5-60. The extension header for routing.

The first 4 bytes of the routing extension header contain four 1-byte integers: the next header type, the routing type (currently 0), the number of addresses present in this header (1 to 24), and the index of the next address to visit. The latter field starts at 0 and is incremented as each address is visited.

Then comes a reserved byte followed by a bit map with bits for each of the 24 potential IPv6 addresses following it. These bits tell whether each address must be visited directly after the one before it (strict source routing), or whether other routers may come in between (loose source routing).

The fragment header deals with fragmentation similarly to the way IPv4 does. The header holds the datagram identifier, fragment number, and a bit telling whether more fragments will follow. In IPv6, unlike in IPv4, only the source host can fragment a packet. Routers along the way may not do this. Although this change is a major philosophical break with the past, it simplifies the routers' work and makes routing go faster. As mentioned above, if a router is confronted with a packet that is too big, it discards the packet and sends an ICMP packet back to the source. This information allows the source host to fragment the packet into smaller pieces using this header and try again.

The authentication header provides a mechanism by which the receiver of a packet can be sure of who sent it. With IPv4, no such guarantee is present. The encrypted security payload makes it possible to encrypt the contents of a packet so that only the intended recipient can read it. These headers use cryptographic techniques to accomplish their missions. We will give brief descriptions below, but readers not already familiar with modern cryptography may not understand the full description now. They should come back after having read Chap. 7 (in particular, Sec. 7.1), which treats cryptographic protocols.

When a sender and receiver wish to communicate securely, they must first agree on one or more secret keys that only they know. How they do this is outside the scope of IPv6. Each of these keys is assigned a unique 32-bit key number. The key numbers are global, so that if Alice is using key 4 to talk to Bob, she cannot also have a key 4 to talk to Carol. Associated with each key number are other parameters, such as key lifetime, and so on.

To send an authenticated message, the sender first constructs a packet consisting of all the IP headers and the payload and then replaces the fields that change underway (e.g., *Hop limit*) with zeros. The packet is then padded out with zeros to a multiple of 16 bytes. Similarly, the secret key to be used is also padded out with zeros to a multiple of 16 bytes. Now a cryptographic checksum is computed on the concatenation of the padded secret key, the padded packet, and the padded secret key again. Users may define their own cryptographic checksum algorithms, but cryptographically unsophisticated users should use the default algorithm, MD5.

Now we come to the role of the authentication header. Basically, it contains three parts. The first part consists of 4 bytes holding the next header number, the length of the authentication header, and 16 zero bits. Then comes the 32-bit key number. Finally, the MD5 (or other) checksum is included.

The receiver then uses the key number to find the secret key. The padded version of it is then prepended and appended to the padded payload, the variable header fields are zeroed out, and the checksum computed. If it agrees with the checksum included in the authentication header, the receiver can be sure that the packet came from the sender with whom the secret key is shared and also be sure that the packet was not tampered with underway. The properties of MD5 make it computationally infeasible for an intruder to forge the sender's identity or modify the packet in a way that escapes detection.

It is important to note that the payload of an authenticated packet is sent unencrypted. Any router along the way can read what it says. For many applications, secrecy is not really important, just authentication. For example, if a user instructs his bank to pay his telephone bill, there is probably no real need for secrecy, but there is a very real need for the bank to be absolutely sure it knows who sent the packet containing the payment order.

For packets that must be sent secretly, the encrypted security payload extension header is used. It starts out with a 32-bit key number, followed by the encrypted payload. The encryption algorithm is up to the sender and receiver, but DES in cipher block chaining mode is the default. When DES-CBC is used, the payload field starts out with the initialization vector (a multiple of 4 bytes), then the payload, then padding out to multiple of 8 bytes. If both encryption and authentication are desired, both headers are needed.

The destination options header is intended for fields that need only be interpreted at the destination host. In the initial version of IPv6, the only options defined are null options for padding this header out to a multiple of 8 bytes, so initially it will not be used. It was included to make sure that new routing and host software can handle it, in case someone thinks of a destination option some day.

Controversies

Given the open design process and the strongly-held opinions of many of the people involved, it should come as no surprise that many choices made for IPv6 were highly controversial. We will summarize a few of these below. For all the gory details, see (Huitema, 1996).

We have already mentioned the argument about the address length. The result was a compromise: 16-byte fixed-length addresses.

Another fight developed over the length of the *Hop limit* field. One camp felt strongly that limiting the maximum number of hops to 255 (implicit in using an 8-bit field) was a gross mistake. After all, paths of 32 hops are common now, and 10 years from now much longer paths may be common. These people argued that using a huge address size was farsighted but using a tiny hop count was shortsighted. In their view, the greatest sin a computer scientist can commit is to provide too few bits somewhere.

The response was that arguments could be made to increase every field, leading to a bloated header. Also, the function of the *Hop limit* field is to keep packets from wandering around for a long time and 65,535 hops is far too long. Finally, as the Internet grows, more and more long-distance links will be built, making it possible to get from any country to any other country in half a dozen hops at most. If it takes more than 125 hops to get from the source and destination to their respective international gateways, something is wrong with the national backbones. The 8-bitters won this one.

Another hot potato was the maximum packet size. The supercomputer community wanted packets in excess of 64 KB. When a supercomputer gets started transferring, it really means business and does not want to be interrupted every 64 KB. The argument against large packets is that if a 1-MB packet hits a 1.5-Mbps T1 line, that packet will tie the line up for over 5 seconds, producing a very noticeable delay for interactive users sharing the line. A compromise was reached here: normal packets are limited to 64 KB, but the hop-by-hop extension header can be used to permit jumbograms.

A third hot topic was removing the IPv4 checksum. Some people likened this move to removing the brakes from a car. Doing so makes the car lighter so it can go faster, but if an unexpected event happens, you have a problem.

The argument against checksums was that any application that really cares about data integrity has to have a transport layer checksum anyway, so having another one in IP (in addition to the data link layer checksum) is overkill. Furthermore, experience showed that computing the IP checksum was a major expense in IPv4. The antichecksum camp won this one, and IPv6 does not have a checksum.

Mobile hosts were also a point of contention. If a portable computer flies halfway around the world, can it continue operating at the destination with the same IPv6 address, or does it have to use a scheme with home agents and foreign agents? Mobile hosts also introduce asymmetries into the routing system. It may well be the case that a small mobile computer can easily hear the powerful signal put out by a large stationary router, but the stationary router cannot hear the feeble signal put out by the mobile host. Consequently, some people wanted to build explicit support for mobile hosts into IPv6. That effort failed when no consensus could be found for any specific proposal.

Probably the biggest battle was about security. Everyone agreed it was needed. The war was about where and how. First where. The argument for putting it in the network layer is that it then becomes a standard service that all applications can use without any advance planning. The argument against it is that really secure applications generally want nothing less than end-to-end encryption, where the source application does the encryption and the destination application undoes it. With anything less, the user is at the mercy of potentially buggy network layer implementations over which he has no control. The response to this argument is that these applications can just refrain from using the IP security features and do the job themselves. The rejoinder to that is that the people who do not trust the network to do it right, do not want to have to pay the price of slow, bulky IP implementations that have this capability, even if it is disabled.

Another aspect of where to put security relates to the fact that many (but not all) countries have stringent export laws concerning cryptography. Some, notably France and Iraq, also greatly restrict its use domestically, so that people cannot have secrets from the police. As a result, any IP implementation that used a cryptographic system strong enough to be of much value could not be exported from the United States (and many other countries) to customers worldwide. Having to maintain two sets of software, one for domestic use and one for export, is something most computer vendors vigorously oppose.

One potential solution is for all vendors to move their cryptography shops to a country that does not regulate cryptography, such as Finland or Switzerland. Strong cryptographic software could be designed and manufactured there and then shipped legally to all countries except France and Iraq. The problem with this approach is that designing part of the router software in one country and part in another can lead to integration problems.

The final controversy concerning security relates to the choice of the default algorithms that all implementations must support. While MD5 was thought to be relatively secure, recent advances in cryptography may weaken it. No serious cryptographer believes that DES is secure against attacks by major governments, but it is probably good enough to foil even the most precocious 12-year-olds for the time being. The compromise was thus to mandate security in IPv6, use a state-of-the-art checksum algorithm for good authentication and a weakish algorithm for secrecy but give users the option of replacing these algorithms with their own.

One point on which there was no controversy is that no one expects the IPv4 Internet to be turned off on a Sunday morning and come back up as an IPv6

Internet Monday morning. Instead, isolated "islands" of IPv6 will be converted, initially communicating via tunnels. As the IPv6 islands grow, they will merge into bigger islands. Eventually, all the islands will merge, and the Internet will be fully converted. Given the massive investment in IPv4 routers currently deployed, the conversion process will probably take a decade. For this reason, an enormous amount of effort has gone into making sure that this transition will be as painless as possible.

5.6. THE NETWORK LAYER IN ATM NETWORKS

The layers of the ATM model (see Fig. 1-30) do not map onto the OSI layers especially well, which leads to ambiguities. The OSI data link layer deals with framing and transfer protocols between two machines on the same physical wire (or fiber). Data link layer protocols are single-hop protocols. They do not deal with end-to-end connections because switching and routing do not occur in the data link layer. About this there is no doubt.

The lowest layer that goes from source to destination, and thus involves routing and switching (i.e., is multihop), is the network layer. The ATM layer deals with moving cells from source to destination and definitely involves routing algorithms and protocols within the ATM switches. It also deals with global addressing. Thus functionally, the ATM layer performs the work expected of the network layer. The ATM layer is not guaranteed to be 100 percent reliable, but that is not required for a network layer protocol.

Also, the ATM layer resembles layer 3 of X.25, which everyone agrees is a network layer protocol. Depending on bit settings, the X.25 network layer protocol may or may not be reliable, but most implementations treat it as unreliable. Because the ATM layer has the functionality expected of the network layer, does not have the functionality expected of the data link layer, and is quite similar to existing network layer protocols, we will discuss the ATM layer in this chapter.

Confusion arises because many people in the ATM community regard the ATM layer as a data link layer, or when doing LAN emulation, even a physical layer. Many people in the Internet community also regard it as a data link layer because they want to put IP on top of it, and making the ATM layer a data link layer fits well with this idea. (Although following through with this line of reasoning, to the Internet community, *all* networks operate at the data link layer, no matter what their physical characteristics are.)

The only problem is that the ATM layer does not have the characteristics of a data link layer protocol: a single-hop protocol used by machines at the opposite ends of a wire, such as protocols 1 through 6 in Chap. 3. It has the characteristics of a network layer protocol: end-to-end virtual circuits, switching, and routing.

The author is reminded of an old riddle:

Q: How many legs would a mule have if you called the tail a leg?
A: Four. *Calling* the tail a leg does not *make* it a leg.

Suffice it to say, the reader is warned of controversy here, combined with a major dose of raw emotion.

The ATM layer is connection oriented, both in terms of the service it offers and the way it operates internally. The basic element of the ATM layer is the virtual circuit (officially called a **virtual channel**). A virtual circuit is normally a connection from one source to one destination, although multicast connections are also permitted. Virtual circuits are unidirectional, but a pair of circuits can be created at the same time. Both parts of the pair are addressed by the same identifier, so effectively a virtual circuit is full duplex. However, the channel capacity and other properties may be different in the two directions and may even be zero for one of them.

The ATM layer is unusual for a connection-oriented protocol in that it does not provide any acknowledgements. The reason for this design is that ATM was designed for use on fiber optic networks, which are highly reliable. It was thought adequate to leave error control to higher layers. After all, sending acknowledgements in the data link or network layer is really only an optimization. It is always sufficient for the transport layer to send a message and then send the entire message again if it is not acknowledged on time.

Furthermore, ATM networks are often used for real-time traffic, such as audio and video. For this kind of traffic, retransmitting an occasional bad cell is worse than just ignoring it.

Despite its lack of acknowledgements, the ATM layer does provide one hard guarantee: cells sent along a virtual circuit will never arrive out of order. The ATM subnet is permitted to discard cells if congestion occurs but under no conditions may it reorder the cells sent on a single virtual circuit. No ordering guarantees are given about cells sent on *different* virtual circuits, however. For example, if a host sends a cell on virtual circuit 10 and later sends a cell on virtual circuit 20 to the same destination, the second cell may arrive first. If the two cells had been sent on the same virtual circuit, the first one sent would always be the first one to arrive.

The ATM layer supports a two-level connection hierarchy that is visible to the transport layer. Along any transmission path from a given source to a given destination, a group of virtual circuits can be grouped together into what is called a **virtual path**, as depicted in Fig. 5-61. Conceptually, a virtual path is like a bundle of twisted copper pairs: when it is rerouted, all the pairs (virtual circuits) are rerouted together. We will consider the implications of this two-level design in detail later.

5.6.1. Cell Formats

In the ATM layer, two interfaces are distinguished: the **UNI** (**User-Network Interface**) and the **NNI** (**Network-Network Interface**). The former defines the boundary between a host and an ATM network (in many cases, between the

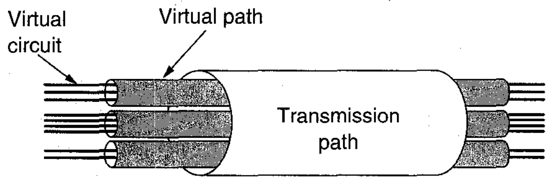


Fig. 5-61. A transmission path can hold multiple virtual paths, each of which can hold multiple virtual circuits.

customer and the carrier). The latter applies to the line between two ATM switches (the ATM term for routers).

In both cases the cells consist of a 5-byte header followed by a 48-byte payload, but the two headers are slightly different. The headers, as defined by the ATM Forum, are illustrated in Fig. 5-62. Cells are transmitted leftmost byte first and leftmost bit within a byte first.

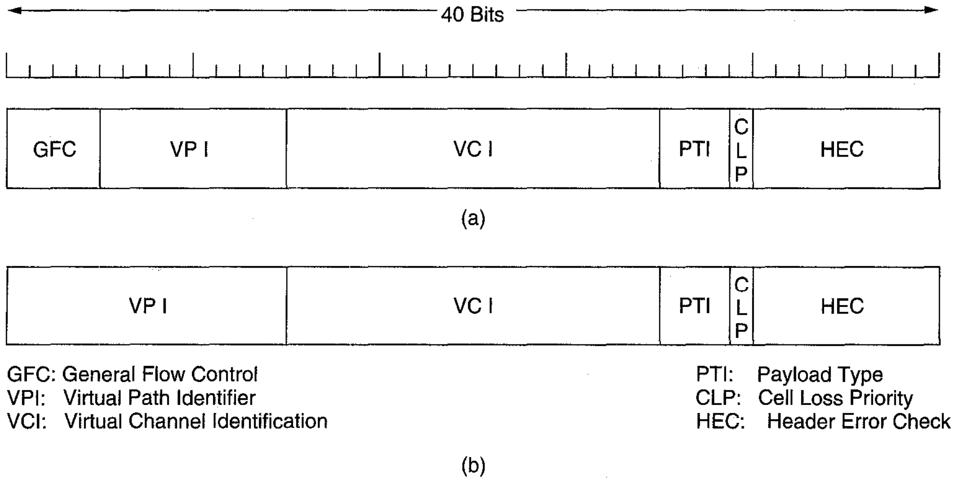


Fig. 5-62. (a) The ATM layer header at the UNI. (b) The ATM layer header at the NNI.

The *GFC* field is present only in cells between a host and the network. It is overwritten by the first switch it reaches, so it does not have end-to-end significance and is not delivered to the destination. It was originally conceived as perhaps having some utility for flow control or priority between hosts and the networks, but no values are defined for it and the network ignores it. The best way to think of it is as a bug in the standard.

The *VPI* field is a small integer selecting a particular virtual path (see Fig. 5-61). Similarly, the *VCI* field selects a particular virtual circuit within the chosen virtual path. Since the *VPI* field has 8 bits (at the UNI) and the *VCI* field has 16 bits, theoretically, a host could have up to 256 VC bundles, each containing up to 65,536 virtual circuits. Actually, slightly fewer of each are available because some *VCI*s are reserved for control functions, such as setting up virtual circuits.

The *PTI* field defines the type of payload the cell contains in accordance with the values given in Fig. 5-63. Here the cell types are user supplied, but the congestion information is network supplied. In other words, a cell sent with *PTI* 000 might arrive with 010 to warn the destination of problems underway.

Payload type	Meaning
000	User data cell, no congestion, cell type 0
001	User data cell, no congestion, cell type 1
010	User data cell, congestion experienced, cell type 0
011	User data cell, congestion experienced, cell type 1
100	Maintenance information between adjacent switches
101	Maintenance information between source and destination switches
110	Resource Management cell (used for ABR congestion control)
111	Reserved for future function

Fig. 5-63. Values of the *PTI* field.

The *CLP* bit can be set by a host to differentiate between high-priority traffic and low-priority traffic. If congestion occurs and cells must be discarded, switches first attempt to discard cells with *CLP* set to 1 before throwing out any set to 0.

Finally, the *HEC* field is a checksum over the header. It does not check the payload. A Hamming code on a 40-bit number only requires 5 bits, so with 8 bits a more sophisticated code can be used. The one chosen can correct all single-bit errors and can detect about 90 percent of all multibit errors. Various studies have shown that the vast majority of errors on optical links are single-bit errors.

Following the header comes 48 bytes of payload. Not all 48 bytes are available to the user, however, since some of the AAL protocols put their headers and trailers inside the payload.

The NNI format is the same as the UNI format, except that the *GFC* field is not present and those 4 bits are used to make the *VPI* field 12 bits instead of 8.

5.6.2. Connection Setup

ATM supports both permanent virtual circuits and switched virtual circuits. The former are always present and can be used at will, like leased lines. The latter have to be established each time they are used, like making phone calls. In this section we will describe how switched virtual circuits are established.

Technically, connection setup is not part of the ATM layer but is handled by the control plane (see Fig. 1-30) using a highly-complex ITU protocol called **Q.2931** (Stiller, 1995). Nevertheless, the logical place to handle setting up a network layer connection is in the network layer, and similar network layer protocols do connection setup here, so we will discuss it here.

Several ways are provided for setting up a connection. The normal way is to first acquire a virtual circuit for signaling and use it. To establish such a circuit, cells containing a request are sent on virtual path 0, virtual circuit 5. If successful, a new virtual circuit is opened on which connection setup requests and replies can be sent and received.

The reason for this two-step setup procedure is that this way the bandwidth reserved for virtual circuit 5 (which is barely used at all) can be kept extremely low. Also, an alternative way is provided to set up virtual circuits. Some carriers may allow users to have permanent virtual paths between predefined destinations, or allow them to set these up dynamically. Once a host has a virtual path to some other host, it can allocate virtual circuits on it itself, without the switches being involved.

Virtual circuit establishment uses the six message types listed in Fig. 5-64. Each message occupies one or more cells and contains the message type, length, and parameters. The messages can be sent by a host to the network or can be sent by the network (usually in response to a message from another host) to a host. Various other status and error reporting messages also exist but are not shown here.

Message	Meaning when sent by host	Meaning when sent by network
SETUP	Please establish a circuit	Incoming call
CALL PROCEEDING	I saw the incoming call	Your call request will be attempted
CONNECT	I accept the incoming call	Your call request was accepted
CONNECT ACK	Thanks for accepting	Thanks for making the call
RELEASE	Please terminate the call	The other side has had enough
RELEASE COMPLETE	Ack for RELEASE	Ack for RELEASE

Fig. 5-64. Messages used for connection establishment and release.

The normal procedure for establishing a call is for a host to send a SETUP message on a special virtual circuit. The network then responds with CALL PROCEEDING to acknowledge receipt of the request. As the SETUP message propagates toward the destination, it is acknowledged at each hop by CALL PROCEEDING.

When the SETUP message finally arrives, the destination host can respond with CONNECT to accept the call. The network then sends a CONNECT ACK message to indicate that it has received the CONNECT message. As the CONNECT message propagates back toward the originator, each switch receiving it acknowledges it with a CONNECT ACK message. This sequence of events is shown in Fig. 5-65(a).

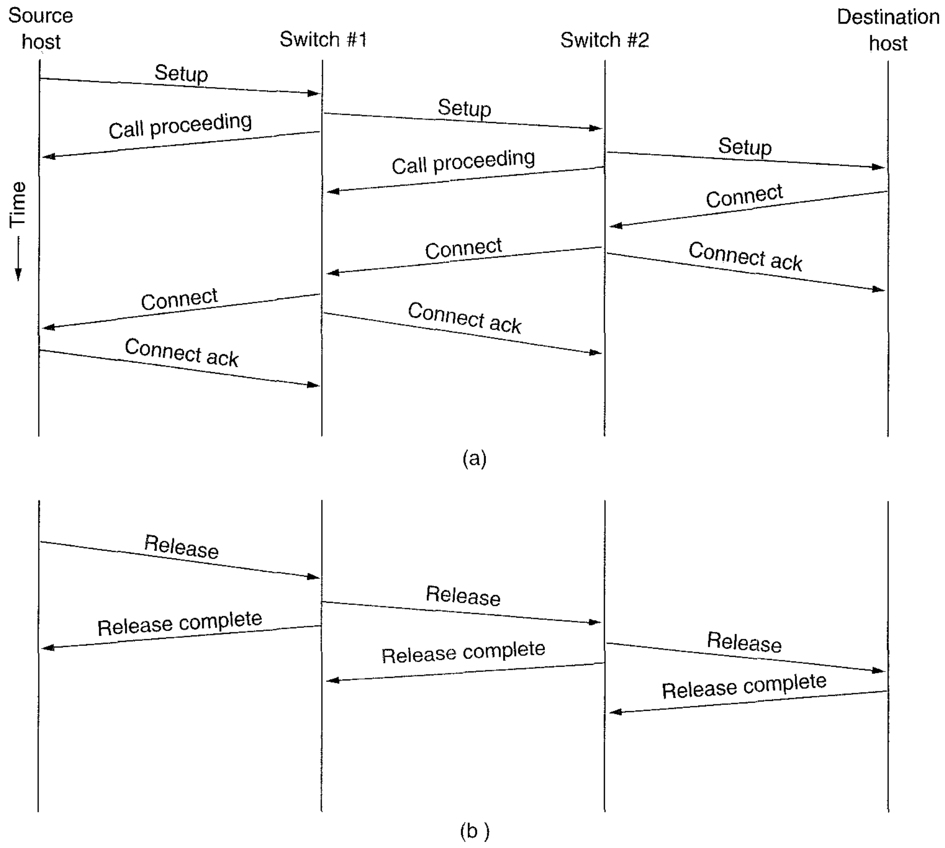


Fig. 5-65. (a) Connection setup in an ATM network. (b) Connection release.

The sequence for terminating a virtual circuit is simple. The host wishing to hang up just sends a RELEASE message, which propagates to the other end and causes the circuit to be released. Each hop along the way, the message is acknowledged, as shown in Fig. 5-65(b).

ATM networks allow multicast channels to be set up. A multicast channel has one sender and more than one receiver. These are constructed by setting up a connection to one of the destinations in the usual way. Then the ADD PARTY message is sent to attach a second destination to the virtual circuit returned by the previous call. Additional ADD PARTY messages can be sent afterward to increase the size of the multicast group.

In order to set up a connection to a destination, it is necessary to specify which destination, by including its address in the SETUP message. ATM addresses come in three forms. The first is 20 bytes long and is based on OSI addresses. The first byte indicates which of three formats the address is in. In the first format, bytes 2 and 3 specify a country, and byte 4 gives the format of the rest of the address, which contains a 3-byte authority, a 2-byte domain, a 2-byte area, and a 6-byte address, plus some other items. In the second format, bytes 2 and 3 designate an international organization instead of a country. The rest of the address is the same as in format 1. Alternatively, a older form of addressing (CCITT E.164) using 15-digit decimal ISDN telephone numbers is also permitted.

5.6.3. Routing and Switching

When a virtual circuit is set up, the SETUP message wends its way through the network from source to destination. The routing algorithm determines the path taken by this message, and thus by the virtual circuit. The ATM standard does not specify any particular routing algorithm, so the carrier is free to choose among the algorithms discussed earlier in this chapter, or to use a different one.

Experience with previous connection-oriented networks, such as X.25, have shown that a considerable amount of computing power in the switches can be expended determining how to convert the virtual circuit information in each cell to the choice of output line. The ATM designers wanted to avoid this fate, so the ATM layer has been designed to make efficient routing possible. In particular, the idea was to route on the *VPI* field, but not the *VCI* field, except at the final hop in each direction, when cells are sent between a switch and a host. Between two switches, only the virtual path was to be used.

Using only the *VPI*s between interior switches has several advantages. To start with, once a virtual path has been established from a source to a destination, any additional virtual circuits along that path can just follow the existing path. No new routing decisions have to be made. It is as though a bundle of twisted pairs has already been pulled from the source to the destination. Setting up a new connection merely requires allocating one of the unused pairs.

Second, routing of individual cells is easier when all virtual circuits for a given path are always in the same bundle. The routing decision only involves looking at a 12-bit number, not a 12-bit number and a 16-bit number. We will describe how cell switching is done below, but even without going into the details, it should be clear that indexing into a table of 2^{12} entries is feasible whereas indexing into a table of 2^{28} entries is not.

Third, basing all routing on virtual paths makes it easier to switch a whole group of virtual circuits. Consider, for example, the hypothetical U.S. ATM backbone illustrated in Fig. 5-66. Normally, virtual circuits from NY to SF pass through Omaha and Denver. However, suppose a disturbance occurs on the Omaha-Denver line. By rerouting the Omaha-Denver virtual path to LA and then SF, all the virtual circuits (potentially up to 65,535 of them) can be switched in one operation instead of potentially thousands of operations.

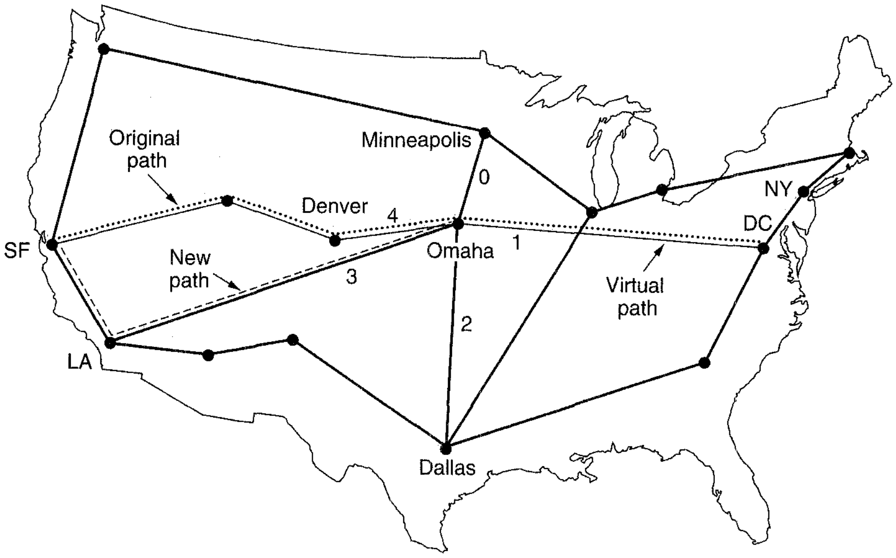


Fig. 5-66. Rerouting a virtual path reroutes all of its virtual circuits.

Finally, virtual paths make it easier for carriers to offer closed user groups (private networks) to corporate customers. A company can set up a network of permanent virtual paths among its various offices, and then allocate virtual circuits within these paths on demand. No calls can come into the private network from outside and no calls can leave the private network, except via special gateways. Many companies like this kind of security.

Whether switches will actually use the *VPI* field for routing as planned, or will, in fact, use the combination of the *VPI* and *VCI* fields (thus negating all the advantages just discussed) remains to be seen. Initial evidence from the field is not encouraging.

Let us now see how cells could be routed within an interior switch (one that is attached only to other switches and not to hosts). To make matters concrete, let us consider the Omaha switch of Fig. 5-66. For each of its five incoming lines, it has a table, *vpi_table*, indexed by incoming *VPI* that tells which of the five outgoing lines to use and what VPI to put in outgoing cells. Let us assume the five lines are numbered from 0 to 4, clockwise starting at Minneapolis. For each outgoing line, the switch maintains a bit map telling which *VPI*s are currently in use on that line.

When the switch is booted, all the entries in all the *vpi_table* structures are marked as not in use. Similarly, all the bit maps are marked to indicate that all *VPI*s are available (except the reserved ones). Now suppose calls come as shown in Fig. 5-67.

As each virtual path (and virtual circuit) is set up, entries are made in the tables. We will assume the virtual circuits are full duplex, so that each one set up

Source	Incoming line	Incoming VPI	Destination	Outgoing line	Outgoing VPI	Path:
NY	1	1	SF	4	1	New
NY	1	2	Denver	4	2	New
LA	3	1	Minneapolis	0	1	New
DC	1	3	LA	3	2	New
NY	1	1	SF	4	1	Old
SF	4	3	DC	1	4	New
DC	1	5	SF	4	4	New
NY	1	2	Denver	4	2	Old
SF	4	5	Minneapolis	0	2	New
NY	1	1	SF	4	1	Old

Fig. 5-67. Some routes through the Omaha switch of Fig. 5-66.

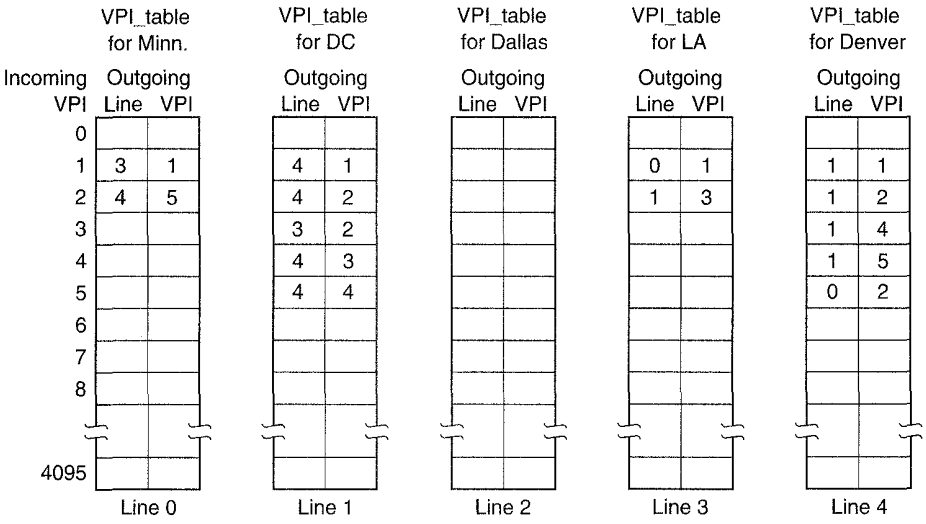
results in two entries, one for the forward traffic from the source and one for the reverse traffic from the destination.

The tables corresponding to the routes of Fig. 5-67 are shown in Fig. 5-68. For example, the first call generates the (4, 1) entry for *VPI* 1 in the DC table because it refers to cells coming in on line 1 with *VPI* 1 and going to SF. However, an entry is also made in the Denver table for *VPI* 1 showing that cells coming in from Denver with *VPI* 1 should go out on line 1 with *VPI* 1. These are cells traveling the other way (from SF to NY) on this virtual path. Note that in some cases two or three virtual circuits are sharing a common path. No new table entries are needed for additional virtual circuits connecting a source and destination that already have a path assigned.

Now we can explain how cells are processed inside a switch. Suppose that a cell arrives on line 1 (DC) with *VPI* 3. The switch hardware or software uses the 3 as an index into the table for line 1 and sees that the cell should go out on line 3 (LA) with *VPI* 2. It overwrites the *VPI* field with a 2 and puts the outgoing line number, 3, somewhere in the cell, for example, in the *HEC* field, since that has to be recomputed later anyway.

Now the question is how to get the cell from its current input buffer to line 3. However, this issue (routing within a switch) was discussed in detail in Chap. 2, and we saw how it was done in knockout and Batcher-banyan switches.

At this point it is straightforward to see how an entire bundle of virtual circuits can be rerouted, as is done in Fig. 5-66. By changing the entry for *VPI* 1 in the DC table from (4, 1) to (3, 3), cells from NY headed for SF will be diverted to LA. Of course, the LA switch has to be informed of this event, so the switch has



	VPI_table for Minn.		VPI_table for DC		VPI_table for Dallas		VPI_table for LA		VPI_table for Denver	
Incoming VPI	Outgoing Line	VPI	Outgoing Line	VPI	Outgoing Line	VPI	Outgoing Line	VPI	Outgoing Line	VPI
0										
1	3	1	4	1			0	1	1	1
2	4	5	4	2			1	3	1	2
3			3	2					1	4
4			4	3					1	5
5			4	4					0	2
6										
7										
8										
4095										
	Line 0		Line 1		Line 2		Line 3		Line 4	

Fig. 5-68. The table entries for the routes of Fig. 5-67.

to generate and send a SETUP message to LA to establish the new path with *VPI* 3. Once this path has been set up, all the virtual circuits from NY to SF are now rerouted via LA, even if there are thousands of them. If virtual paths did not exist, each virtual circuit would have its own table entry and would have to be rerouted separately.

It is worth pointing out explicitly that the discussion above is about ATM in WANs. In a LAN, matters are much simpler. For example, a single virtual path can be used for all virtual circuits.

5.6.4. Service Categories

After a fair amount of trial and error, by version 4.0 of the ATM specification, it was becoming clear what kinds of traffic ATM networks were carrying and what kind of services their customers wanted. Consequently, the standard was modified to explicitly list the service categories commonly used, in order to allow equipment vendors to optimize their adaptor boards and switches for some or all of these categories. The service categories chosen as being important are listed in Fig. 5-69.

The **CBR (Constant Bit Rate)** class is intended to emulate a copper wire or optical fiber (only at much greater expense). Bits are put on one end and they come off the other end. No error checking, flow control, or other processing is done. Nevertheless, this class is essential to making a smooth transition between

Class	Description	Example
CBR	Constant bit rate	T1 circuit
RT-VBR	Variable bit rate: real time	Real-time videoconferencing
NRT-VBR	Variable bit rate: non-real time	Multimedia email
ABR	Available bit rate	Browsing the Web
UBR	Unspecified bit rate	Background file transfer

Fig. 5-69. The ATM service categories.

the current telephone system and future B-ISDN systems, since voice-grade PCM channels, T1 circuits, and most of the rest of the telephone system use constant-rate, synchronous bit transmission. With the CBR class, all of this traffic can be carried directly by an ATM system. CBR is also suited to all other interactive (i.e., real-time) audio and video streams.

The next class, **VBR (Variable Bit Rate)**, is divided into two subclasses, for real time and non-real time, respectively. RT-VBR is intended for services that have variable bit rates combined with stringent real-time requirements, such as interactive compressed video (e.g., videoconferencing). Due to the way MPEG and other compression schemes work, with a complete base frame followed by a series of differences between the current frame and the base frame, the transmission rate varies strongly in time (Pancha and El Zarki, 1994). Despite this variation, it is essential that the ATM network not introduce any jitter in the cell arrival pattern, as this will cause the display to appear jerky. In other words, both the average cell delay and the variation in cell delay must be tightly controlled. On the other hand, an occasional lost bit or cell here is tolerable and is best just ignored.

The other VBR subclass is for traffic where timely delivery is important but a certain amount of jitter can be tolerated by the application. For example, multimedia email is typically spooled to the receiver's local disk before being displayed, so any variation in cell delivery times will be eliminated before the email is viewed.

The **ABR (Available Bit Rate)** service category is designed for bursty traffic whose bandwidth range is known roughly. A typical example might be for use in a company that currently connects its offices by a collection of leased lines. Typically, the company has a choice of putting in enough capacity to handle the peak load, which means that some lines are idle part of the day, or putting in just enough capacity for the minimum load, which leads to congestion during the busiest part of the day.

Using ABR service avoids having to make a long term commitment to a fixed bandwidth. With ABR it is possible to say, for example, that the capacity between two points must always be 5 Mbps, but might have peaks up to 10 Mbps.

The system will then guarantee 5 Mbps all the time, and do its best to provide 10 Mbps when needed, but with no promises.

ABR is the only service category in which the network provides rate feedback to the sender, asking it to slow down when congestion occurs. Assuming that the sender complies with such requests, cell loss for ABR traffic is expected to be low. Traveling ABR is a little like flying standby: if there are seats left over (excess capacity), standby passengers are transported without delay. If there is insufficient capacity, they have to wait (unless some of the minimum bandwidth is available).

Finally, we come to **UBR (Unspecified Bit Rate)**, which makes no promises and gives no feedback about congestion. This category is well suited to sending IP packets, since IP also makes no promises about delivery. All UBR cells are accepted, and if there is capacity left over, they will also be delivered. If congestion occurs, UBR cells will be discarded, with no feedback to the sender and no expectation that the sender slows down.

To continue our standby analogy, with UBR, all standby passengers get to board, but if halfway to the destination the pilot sees that fuel is running low, standby passengers are unceremoniously pushed through the emergency exit. To make UBR attractive, carriers are likely to make it cheaper than the other classes. For applications that have no delivery constraints and want to do their own error control and flow control anyway, UBR is a perfectly reasonably choice. File transfer, email, and USENET news are all potential candidates for UBR service because none of these applications have real-time characteristics.

The properties of the various service categories are summarized in Fig. 5-70.

Service characteristic	CBR	RT-VBR	NRT-VBR	ABR	UBR
Bandwidth guarantee	Yes	Yes	Yes	Optional	No
Suitable for real-time traffic	Yes	Yes	No	No	No
Suitable for bursty traffic	No	No	Yes	Yes	Yes
Feedback about congestion	No	No	No	Yes	No

Fig. 5-70. Characteristics of the ATM service categories.

5.6.5. Quality of Service

Quality of service is an important issue for ATM networks, in part because they are used for real-time traffic, such as audio and video. When a virtual circuit is established, both the transport layer (typically a process in the host machine, the "customer") and the ATM network layer (e.g., a network operator, the "carrier") must agree on a contract defining the service. In the case of a public network, this contract may have legal implications. For example, if the carrier agrees not to lose more than one cell per billion and it loses two cells per billion, the customer's legal staff may get all excited and start running around yelling "breach of contract."

The contract between the customer and the network has three parts:

1. The traffic to be offered.
2. The service agreed upon.
3. The compliance requirements.

It is worth noting that the contract may be different for each direction. For a video-on-demand application, the required bandwidth from the user's remote control to the video server might be 1200 bps. In the other direction it might be 5 Mbps. It should be noted that if the customer and the carrier cannot agree on terms, or the carrier is unable to provide the service desired, the virtual circuit will not be set up.

The first part of the contract is the **traffic descriptor**. It characterizes the load to be offered. The second part of the contract specifies the quality of service desired by the customer and accepted by the carrier. Both the load and the service must be formulated in terms of measurable quantities, so compliance can be objectively determined. Merely saying "moderate load" or "good service" will not do.

To make it possible to have concrete traffic contracts, the ATM standard defines a number of **QoS** (**Quality of Service**) parameters whose values the customer and carrier can negotiate. For each quality of service parameter, the worst case performance for each parameter is specified, and the carrier is required to meet or exceed it. In some cases, the parameter is a minimum; in others it is a maximum. Again here, the quality of service is specified separately for each direction. Some of the more important ones are listed in Fig. 5-71, but not all of them are applicable to all service categories.

The first three parameters specify how fast the user wants to send. **PCR** (**Peak Cell Rate**) is the maximum rate at which the sender is planning to send cells. This parameter may be lower than what the bandwidth of the line permits. If the sender is planning to push out a cell every 4 μsec, its *PCR* is 250,000 cells/sec, even though the actual cell transmission time may be 2.7 μsec.

SCR (**Sustained Cell Rate**) is the expected or required cell rate averaged over a long time interval. For CBR traffic, *SCR* will be equal to *PCR*, but for all the other service categories, it will be substantially lower. The *PCR/SCR* ratio is one measure of the burstiness of the traffic.

MCR (**Minimum Cell Rate**) is the minimum number of cells/sec that the customer considers acceptable. If the carrier is unable to guarantee to provide this much bandwidth it must reject the connection. When ABR service is requested, then the actual bandwidth used must lie between *MCR* and *PCR*, but it may vary

Parameter	Acronym	Meaning
Peak cell rate	PCR	Maximum rate at which cells will be sent
Sustained cell rate	SCR	The long-term average cell rate
Minimum cell rate	MCR	The minimum acceptable cell rate
Cell delay variation tolerance	CDVT	The maximum acceptable cell jitter
Cell loss ratio	CLR	Fraction of cells lost or delivered too late
Cell transfer delay	CTD	How long delivery takes (mean and maximum)
Cell delay variation	CDV	The variance in cell delivery times
Cell error rate	CER	Fraction of cells delivered without error
Severely-errored cell block ratio	SECBR	Fraction of blocks garbled
Cell misinsertion rate	CMR	Fraction of cells delivered to wrong destination

Fig. 5-71. Some of the quality of service parameters.

dynamically during the lifetime of the connection. If the customer and carrier agree to setting MCR to 0, then ABR service becomes similar to UBR service.

CVDT (**Cell Variation Delay Tolerance**) tells how much variation will be present in cell transmission times. It is specified independently for *PCR* and *SCR*. For a perfect source operating at *PCR*, every cell will appear *exactly* 1/*PCR* after the previous one. No cell will ever be early and no cell will ever be late, not even by a picosecond. For a real source operating at *PCR*, some variation will occur in cell transmission times. The question is: How much variation is acceptable? Can a cell be 1 nsec early? How about 30 seconds? *CDVT* controls the amount of variability acceptable using a leaky bucket algorithm to be described shortly.

The next three parameters describe characteristics of the network and are measured at the receiver. All three are negotiable. **CLR** (**Cell Loss Ratio**) is straightforward. It measures the fraction of the transmitted cells that are not delivered at all or are delivered so late as to be useless (e.g., for real-time traffic). **CTD** (**Cell Transfer Delay**) is the average transit time from source to destination. **CDV** (**Cell Delay Variation**) measures how uniformly the cells are delivered.

The model for *CDT* and *CDV* is shown in Fig. 5-72. Here we see the probability of a cell taking time t to arrive, as a function of t. For a given source, destination, and route through the intermediate switches, some minimum delay always exists due to propagation and switching time. However, not all cells make it in the minimum time; the probability density function usually has a long tail. By choosing a value of *CDT*, the customer and the carrier are, in effect, agreeing, on how late a cell can be delivered and still count as a correctly delivered cell. Normally, *CDV* will be chosen so that, α, the fraction of cells that are rejected for being too late will be on the order of 10^{-10} or less. *CDV* measures the spread in arrival times. For real-time traffic, this parameter is often more important than *CDT*.

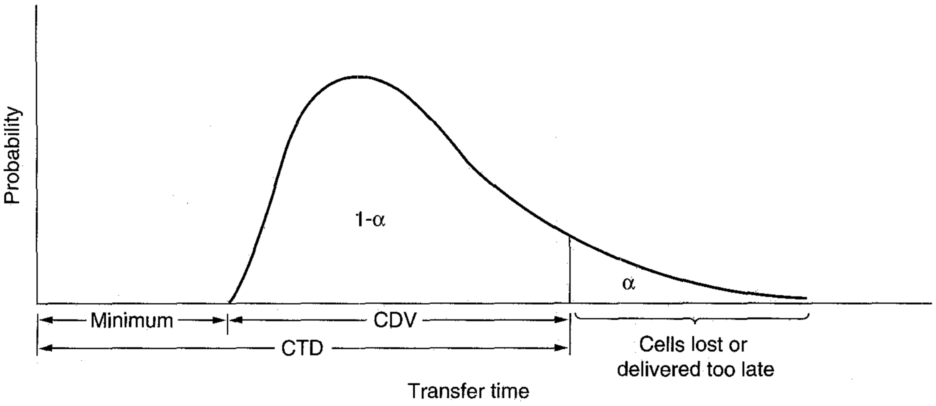


Fig. 5-72. The probability density function for cell arrival times.

The last three QoS parameters specify characteristics of the network. They are generally not negotiable. **CER** (**Cell Error Ratio**) is the fraction of cells that are delivered with one or more bits wrong. **SECBR** (**Severely-Errored Cell Block Ratio**) is the fraction of *N*-cell blocks of which *M* or more cells contain an error. Finally, **CMR** (**Cell Misinsertion Rate**) is the number of cells/sec that are delivered to the wrong destination on account of an undetected error in the header.

The third part of the traffic contract tells what constitutes obeying the rules. If the customer sends one cell too early, does this void the contract? If the carrier fails to meet one of its quality targets for a period of 1 msec, can the customer sue? Effectively, this part of the contract is negotiated between the parties and says how strictly the first two parts will be enforced.

The ATM and Internet quality of service models differ somewhat, which impacts their respective implementations. The ATM model is based strictly on connections, whereas the Internet model uses datagrams plus flows (e.g., RSVP). A comparison of these two models is given in (Crowcroft et al., 1995).

5.6.6. Traffic Shaping and Policing

The mechanism for using and enforcing the quality of service parameters is based (in part) on a specific algorithm, the **Generic Cell Rate Algorithm** (**GCRA**), and is illustrated in Fig. 5-73. It works by checking every cell to see if it conforms to the parameters for its virtual circuit.

GCRA has two parameters. These specify the maximum allowed arrival rate (*PCR*) and the amount of variation herein that is tolerable (*CDVT*). The

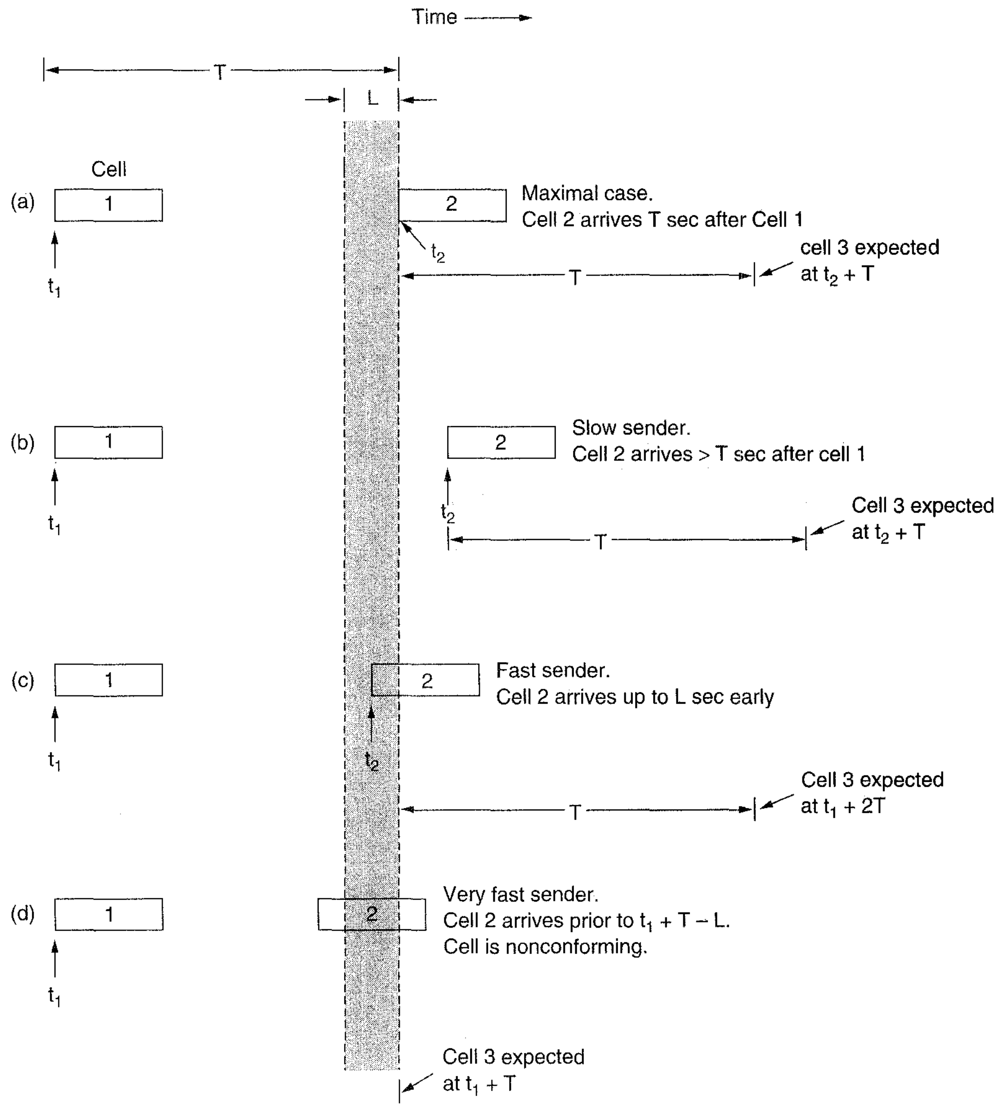


Fig. 5-73. The generic cell rate algorithm.

reciprocal of *PCR*, $T = 1/PCR$, is the minimum cell interarrival time, as shown in Fig. 5-73(a). If the customer agrees not to send more than 100,000 cells/sec, then $T = 10$ μsec. In the maximal case, one cell arrives promptly every 10 μsec. To avoid tiny numbers, we will work in microseconds, but since all the parameters are real numbers, the unit of time does not matter.

A sender is always permitted to space consecutive cells more widely than T, as shown in Fig. 5-73(b). Any cell arriving more than T μsec after the previous one is conforming.

The problem arises with senders that tend to jump the gun, as in Fig. 5-73(c) and (d). If a cell arrives a little early (at or later than $t_1 + T - L$), it is conforming, but the next cell is still expected at $t_1 + 2T$, (not at $t_2 + T$), to prevent the sender from transmitting every cell L μsec early, and thus increasing the peak cell rate.

If a cell arrives more than L μsec early, it is declared as nonconforming. The treatment of nonconforming cells is up to the carrier. Some carriers may simply discard them; others may keep them, but set the *CLP* bit, to mark them as low priority to allow switches to drop nonconforming cells first in the event of congestion. The use of the *CLP* bit may also be different for the different service categories of Fig. 5-69.

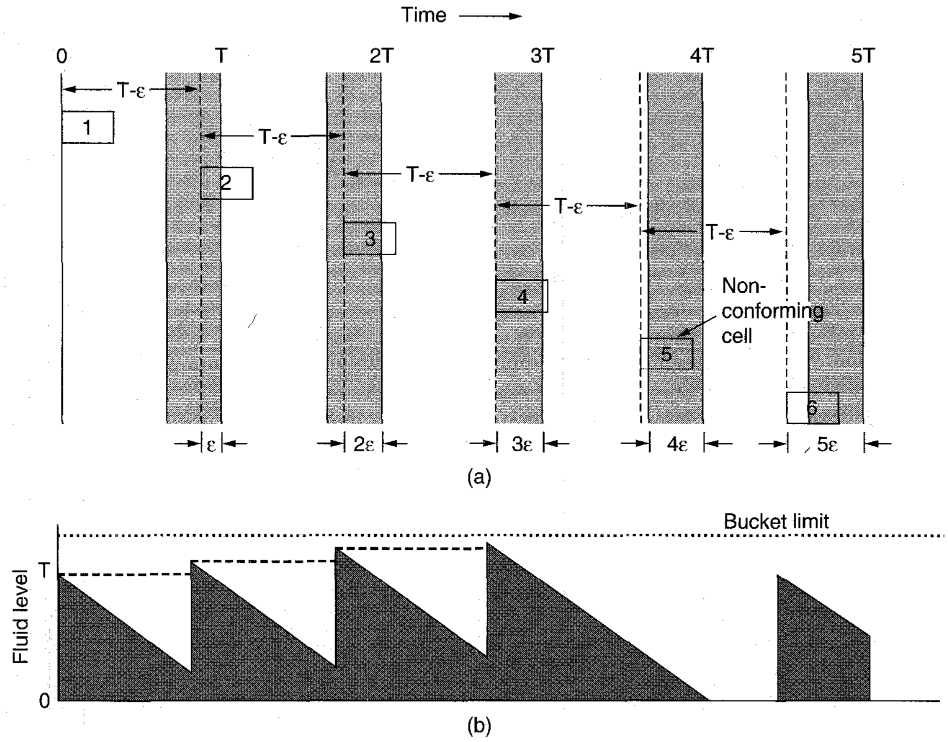


Fig. 5-74. (a) A sender trying to cheat. (b) The same cell arrival pattern, but now viewed in terms of a leaky bucket.

Now let us consider what happens if a sender tries to cheat a little bit, as shown in Fig. 5-74(a). Instead of waiting until time T to send cell 2, the sender transmits it a wee bit early, at $T - \varepsilon$, where, say, $\varepsilon = 0.3L$. This cell is accepted without problems.

Now the sender transmits cell 3, again at $T - \varepsilon$ after the previous cell, that is, at $T - 2\varepsilon$. Again it is accepted. However, every successive cell inches closer and closer to the fatal $T - L$ boundary. In this case, cell 5 arrives at $T - 4\varepsilon$ ($T - 1.2L$) which is too early, so cell 5 is declared nonconforming and is discarded by the network interface.

When viewed in these terms, the GCRA algorithm is called a **virtual scheduling algorithm**. However, viewed differently, it is equivalent to a leaky bucket algorithm, as depicted in Fig. 5-74(b). Imagine that each conforming cell that arrives pours T units of fluid into a leaky bucket. The bucket leaks fluid at a rate of 1 unit/μsec, so that after T μsec it is all gone. If cells arrive precisely every T μsec, each arriving cell will find the bucket (just) emptied, and will refill it with T units of fluid. Thus the fluid level is raised to T when a cell arrives and is reduced linearly until it gets to zero. This situation is illustrated in Fig. 5-74(b) between 0 and T.

Since fluid drains out linearly in time, at a time t after a cell arrives, the amount of its fluid left is $T - t$. At the time cell 2 arrives, at $T - \varepsilon$, there are still ε units of fluid in the bucket. The addition of the new cell raises this value to $T + \varepsilon$. Similarly, at the time cell 3 arrives, 2ε units are left in the bucket so the new cell raises the fluid level to $T + 2\varepsilon$. When cell 4 arrives, it is raised to $T + 3\varepsilon$.

If this goes on indefinitely, some cell is going to raise the level to above the bucket capacity and thus be rejected. To see which one it is, let us now compute what the bucket capacity is. We want the leaky bucket algorithm to give the same result as Fig. 5-74(a), so we want overflow to occur when a cell arrives L μsec early. If the fluid left requires L μsec to drain out, the amount of fluid must be L, since the drain rate is 1 unit/μsec. Thus we want the bucket capacity to be $L + T$ so that any cell arriving more than L μsec early will be rejected due to bucket overflow. In Fig. 5-74(b), when cell 5 arrives, the addition of T units to the 4ε units of fluid already present raises the bucket level to $T + 4\varepsilon$. Since we are using $\varepsilon = 0.3L$ in this example, the bucket would be raised to $T + 1.2L$ by the addition of cell 5, so the cell is rejected, no new fluid is added, and the bucket eventually empties.

For a given T, if we set L very small, the capacity of the bucket will be hardly more than T, so all cells will have to be sent with a very uniform spacing. However, if we now raise L to a value much greater than T, the bucket can hold multiple cells because $T + L \gg T$. This means that the sender can transmit a burst of cells back-to-back at the peak rate and have them still accepted.

We can easily compute the number of conforming cells, N, that can be transmitted back-to-back at the peak cell rate ($PCR = 1/T$). During a burst of N cells, the total amount of fluid added to the bucket is NT because each cell adds T. However, during the maximum burst, fluid drains out of the bucket at a rate of 1 unit per time interval. Let us call the cell transmission time δ time units. Note that $\delta \leq T$ because it is entirely possible for a sender on a 155.52 Mbps line to agree to send no more than 100,000 cells/sec, in which case $\delta = 2.73$ μsec and $T = 10$ μsec. During the burst of N cells, the amount of drainage is $(N - 1)\delta$ because drainage does not start until the first cell has been entirely transmitted.

From these observations, we set that the net increase in fluid in the bucket during the maximum burst is $NT - (N - 1)\delta$. The bucket capacity is $T + L$. Equating these two quantities we get

$$NT - (N - 1)\delta = T + L$$

Solving this equation for N we get

$$N = 1 + \frac{L}{T - \delta}$$

However, if this number is not an integer, it must be rounded downward to an integer to prevent bucket overflow. For example, with PCR = 100,000 cells/sec, $\delta = 2.73$ μsec, and $L = 50$ μsec, seven cells may be sent back-to-back at the 155.52 Mbps rate without filling the bucket. An eighth cell would be nonconforming.

The GCRA is normally specified by giving the parameters T and L. T is just the reciprocal of *PCR*; L is *CDVT*. The GCRA is also used to make sure the mean cell rate does not exceed *SCR* for any substantial period.

In this example we assumed that cells arrive uniformly. In reality, they do not. Nevertheless, the leaky bucket algorithm can also be used here, too. At every cell arrival, a check is made to see if there is room in the bucket for an additional T units of fluid. If there is, the cell is conforming; otherwise it is not.

In addition to providing a rule about which cells are conforming and which ones are not, the GCRA also shapes the traffic to remove some of the burstiness. The smaller *CDVT* is, the greater the smoothing effect, but the greater the chance that cells will be discarded as nonconforming. Some implementations combine the GCRA leaky bucket with a token bucket, to provide additional smoothing.

5.6.7. Congestion Control

Even with traffic shaping, ATM networks do not automatically meet the performance requirements set forth in the traffic contract. For example, congestion at intermediate switches is always a potential problem, especially when over 350,000 cells/sec are pouring in on each line, and a switch can have 100 lines. Consequently, a great deal of thought has gone into the subject of performance and congestion in ATM networks. In this section, we will discuss some of the approaches used. For additional information, see (Eckberg, 1992; Eckberg et al., 1991; Hong and Suda, 1991; Jain, 1995; and Newman, 1994).

ATM networks must deal with both long-term congestion, caused by more traffic coming in than the system can handle, and short-term congestion, caused by burstiness in the traffic. As a result, several different strategies are used together. The most important of these fall into three categories:

1. Admission control.
2. Resource reservation.
3. Rate-based congestion control.

We will now discuss each of these strategies in turn.

Admission Control

In low-speed networks, it is usually adequate to wait for congestion to occur and then react to it by telling the source of the packets to slow down. In high-speed networks, this approach often works poorly, because in the interval between sending the notification and notification arriving at the source, thousands of additional packets may arrive.

Furthermore, many ATM networks have real-time traffic sources that produce data at an intrinsic rate. Telling such a source to slow down may not work (imagine a new digital telephone with a red light on it; when congestion is signaled, the red line comes on and the speaker is required to talk 25 percent slower).

Consequently, ATM networks emphasize preventing congestion from occurring in the first place. However, for CBR, VBR, and UBR traffic, no dynamic congestion control is present at all, so here an ounce of prevention is worth a pound (actually, more like a metric ton) of cure. A major tool for preventing congestion is admission control. When a host wants a new virtual circuit, it must describe the traffic to be offered and the service expected. The network can then check to see if it is possible to handle this connection without adversely affecting existing connections. Multiple potential routes may have to be examined to find one which can do the job. If no route can be located, the call is rejected.

Denying admission should be done fairly. Is it fair that one couch potato zapping through dozens of television programs can wipe out 100 busy beavers trying to read their email? If no controls are applied, a small number of high-bandwidth users can severely affect many low-bandwidth users. To prevent this, users should be divided into classes based on usage. The probability of service denial should be roughly the same for all classes (possibly by giving each class its own resource pool).

Resource Reservation

Closely related to admission control is the technique of reserving resources in advance, usually at call setup time. Since the traffic descriptor gives the peak cell rate, the network has the possibility of reserving enough bandwidth along the path to handle that rate. Bandwidth can be reserved by having the SETUP message earmark bandwidth along each line it traverses, making sure, of course, that the total bandwidth earmarked along a line is less than the capacity of that line. If the SETUP message hits a line that is full, it must backtrack and look for an alternative path.

The traffic descriptor can contain not only the peak bandwidth, but also the average bandwidth. If a host wants, for example, a peak bandwidth of 100,000 cells/sec, but an average bandwidth of only 20,000 cells/sec, in principle, five such circuits could be multiplexed onto the same physical trunk. The trouble is that all five connections could be idle for half an hour, then start blasting away at the peak rate, causing massive cell loss. Since VBR traffic can be statistically multiplexed, problems can occur with this service category. Possible solutions are being studied.

Rate-Based Congestion Control

With CBR and VBR traffic, it is generally not possible for the sender to slow down, even in the event of congestion, due to the inherent real-time or semi-real-time nature of the information source. With UBR, nobody cares; if there are too many cells, the extra ones are just dropped.

However, with ABR traffic, it is possible and reasonable for the network to signal one or more senders and ask them to slow down temporarily until the network can recover. It is in the interest of a sender to comply, since the network can always punish it by throwing out its (excess) cells.

How congestion should be detected, signaled, and controlled for ABR traffic was a hot topic during the development of the ATM standard, with vigorous arguments for various proposed solutions. Let us now briefly look at some of the solutions that were quickly rejected before examining the winner.

In one proposal, whenever a sender wished to send a burst of data, it first had to send a special cell reserving the necessary bandwidth. After the acknowledgement came back, the burst could begin. The advantage here is that congestion never occurs because the required bandwidth is always there when it is needed. The ATM Forum rejected this solution due to the potentially long delay before a host may begin to send.

A second proposal had switches sending back choke cells whenever congestion began to occur. Upon receipt of such a cell, a sender was expected to cut back to half its current cell transmission rate. Various schemes were proposed for getting the rate back up again later when the congestion cleared up. This scheme was rejected because choke cells might get lost in the congestion, and because the scheme seemed unfair to small users. For example, consider a switch getting 100-Mbps streams from each of five users, and one 100-kbps stream from another user. Many committee members felt it was inappropriate to tell the 100 kbps user to give up 50 kbps because he was causing too much congestion.

A third proposal used the fact that packet boundaries are marked by a bit in the last cell. The idea here was to discard cells to relieve the congestion but to do this highly selectively. The switch was to scan the incoming cell stream for the end of a packet and then throw out all the cells in the next packet. Of course, this one packet would be transmitted later, but dropping all k cells in one packet ultimately leads to one packet retransmission, which is far better than dropping k random cells, which might lead to k packet retransmissions. This scheme was rejected on fairness grounds because the next end-of-packet mark seen might not belong to the sender overloading the switch. Also the scheme did not need to be standardized. Any switch vendor is free to pick which cells to discard when congestion occurs.

After much discussion, the battle focused on two contenders, a credit-based solution (Kung and Morris, 1995) and rate-based solution (Bonomi and Fendick, 1995). The credit-based solution was essentially a dynamic sliding window protocol. It required each switch to maintain, per virtual circuit, a credit—effectively the number of buffers reserved for that circuit. As long as each transmitted cell had a buffer waiting for it, congestion could never arise.

The argument against it came from the switch vendors. They did not want to do all the accounting to keep track of the credits and did not want to reserve so many buffers in advance. The amount of overhead and waste required was thought to be too much, so ultimately, the rate-based congestion control scheme was adopted. It works like this.

The basic model is that after every k data cells, each sender transmits a special **RM** (**Resource Management**) cell. This cell travels along the same path as the data cells, but is treated specially by the switches along the way. When it gets to the destination, it is examined, updated, and sent back to the sender. The full path for RM cells is shown in Fig. 5-75.

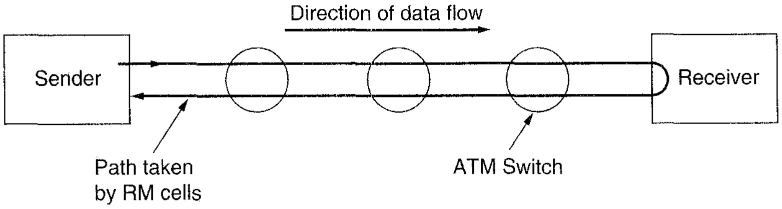


Fig. 5-75. The path taken by RM cells.

In addition, two other congestion control mechanisms are provided. First, overloaded switches can spontaneously generate RM cells and ship them back to the sender. Second, overloaded switches can set the middle *PTI* bit on data cells traveling from the sender to the receiver. Neither of these methods are fully reliable however, since these cells may be lost in the congestion without anyone noticing. In contrast, a lost RM cell will be noticed by the sender when it fails to return within the expected time interval. As an aside, the *CLP* bit is not used for ABR congestion control.

ABR congestion control is based on the idea that each sender has a current rate, **ACR (Actual Cell Rate)** that falls between *MCR* and *PCR*. When congestion occurs, *ACR* is reduced (but not below *MCR*). When congestion is absent, *ACR* is increased (but not above *PCR*). Each RM cell sent contains the rate at which the sender would currently like to transmit, possibly *PCR*, possibly lower. This value is called **ER (Explicit Rate)**. As the RM cell passes through various switches on the way to the receiver, those that are congested may reduce *ER*. No switch may increase it. Reduction can occur either in the forward direction or in the reverse direction. When the sender gets the RM cell back, it can then see what the minimum acceptable rate is according to all the switches along the path. It can then adjust *ACR*, if need be, to bring it into line with what the slowest switch can handle.

The congestion mechanism using the middle *PTI* bit is integrated into the RM cells by having the receiver include this bit (taken from the last data cell) in each RM cell sent back. The bit cannot be taken from the RM cell itself because all RM cells have this bit set all the time, as shown in Fig. 5-63.

The ATM layer is quite complicated. In this chapter, we have highlighted only a portion of the issues. For additional information, see (De Prycker, 1993; McDysan and Spohn, 1995; Minoli and Vitella, 1994; and La Porta et al., 1994). However, the reader should be warned that all these references discuss the ATM 3 standard, not the ATM 4 standard, which was not finalized until 1996.

5.6.8. ATM LANs

As it becomes increasingly obvious that ITU's original goal of replacing the public switched telephone network by an ATM network is going to take a very long time, attention is shifting to the use of ATM technology to connect existing LANs together. In this approach, an ATM network can function either as a LAN, connecting individual hosts, or as a bridge, connecting multiple LANs. Although both concepts are interesting, they raise some challenging issues that we will discuss below. Additional information about ATM LANs can be found in (Chao et al., 1994; Newman, 1994; Truong et al., 1995).

The major problem that must be solved is how to provide connectionless LAN service over a connection-oriented ATM network. One possible solution is to introduce a connectionless server into the network. Every host initially sets up a connection to this server, and sends all packets to it for forwarding. While simple, this solution does not use the full bandwidth of the ATM network, and the connectionless server can easily become a bottleneck.

An alternative approach, proposed by the ATM Forum, is shown in Fig. 5-76. Here every host has a (potential) ATM virtual circuit to every other host. These virtual circuits can be established and released dynamically as needed, or they can be permanent virtual circuits. To send a frame, the source host first encapsulates the packet in the payload field of an ATM AAL message and sends it to the destination, the same way frames are shipped over Ethernets, token rings, and other LANs.

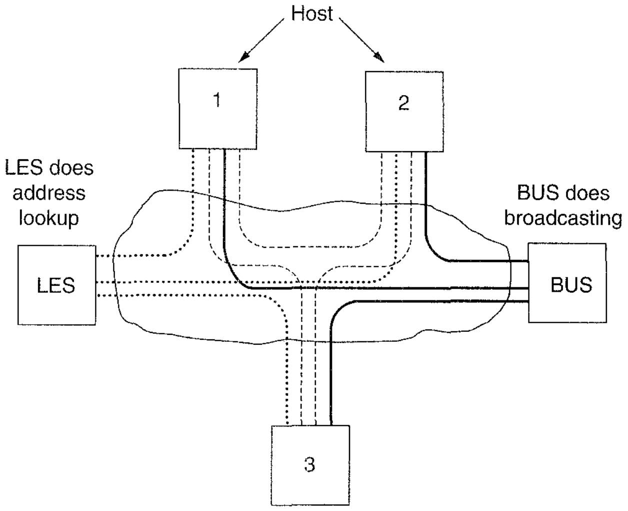


Fig. 5-76. ATM LAN emulation.

The main problem introduced by this scheme is how to tell which IP (or other network layer address) belongs to which virtual circuit. In an 802 LAN, this problem is solved by the ARP protocol, in which a host can broadcast a request such as: "Who has IP address 192.31.20.47?" The host using that address then sends back a point-to-point response, which is cached for later use.

With an ATM LAN, this solution does not work because ATM LANs do not support broadcasting. This problem is solved by introducing a new server, the **LES** (**LAN Emulation Server**). To look up a network layer address (e.g., an IP address), a host sends a packet (e.g., an ARP request) to the LES, which then looks up the corresponding ATM address and returns it to the machine requesting it. This address can then be used to send encapsulated packets to the destination.

However, this solution only solves the host location problem. Some programs use broadcasting or multicasting as an essential part of the application. For these applications, the **BUS** (**Broadcast/Unknown Server**) is introduced. It has connections to all hosts and can simulate broadcasting by sending a packet to all of them, one at a time. Hosts can also speed up delivery of a packet to an unknown host by sending the packet to the BUS for broadcasting and then (in parallel) looking up the address (for future use) using the LES.

A model similar to this one has been adopted by the IETF as the official Internet way to use an ATM network for transporting IP packets. In this model the

LES server is called the **ATMARP** server, but the functionality is essentially the same. Broadcasting and multicasting are not supported in the IETF proposal. The model is described in RFC 1483 and RFC 1577. Another good source of information is (Comer, 1995).

In the IETF method, a set of ATM hosts can be grouped together to form a **logical IP subnet**. Each LIS has its own ATMARP server. In effect, a LIS acts like a virtual LAN. Hosts on the same LIS may exchange IP packets directly, but hosts on different ones are required to go through a router. The reason for having LISes is that every host on a LIS must (potentially) have an open virtual circuit to every other host on its LIS. By restricting the number of hosts per LIS, the number of open virtual circuits can be reduced to a manageable number.

Another use of ATM networks is to use them as bridges to connect existing LANs. In this configuration, only one machine on each LAN needs an ATM connection. Like all transparent bridges, the ATM bridge must listen promiscuously to all LANs to which it is attached, forwarding frames where needed. Since bridges use only MAC addresses (not IP addresses), ATM bridges must build a spanning tree, just as 802 bridges.

In short, while ATM LAN emulation is an interesting idea, there are serious questions about its performance and price, and there is certainly heavy competition from existing LANs and bridges, which are well established and highly optimized. Whether ATM LANs and bridges ever replace 802 LANs and bridges remains to be seen.

5.7. SUMMARY

The network layer provides services to the transport layer. It can be based on either virtual circuits or datagrams. In both cases, its main job is routing packets from the source to the destination. In virtual circuit subnets, a routing decision is made when the virtual circuit is set up. In datagram subnets, it is made on every packet.

Many routing algorithms are used in computer networks. Static algorithms include shortest path routing, flooding, and flow-based routing. Dynamic algorithms include distance vector routing and link state routing. Most actual networks use one of these. Other important routing topics are hierarchical routing, routing for mobile hosts, broadcast routing, and multicast routing.

Subnets can become congested, increasing the delay and lowering the throughput for packets. Network designers attempt to avoid congestion by proper design. Techniques include traffic shaping, flow specifications, and bandwidth reservation. If congestion does occur, it must be dealt with. Choke packets can be sent back, load can be shed, and other methods applied.

Networks differ in various ways, so when multiple networks are connected together problems can occur. Sometimes the problems can be finessed by tunneling a packet through a hostile network, but if the source and destination networks are different, this approach fails. When different networks have different maximum packet sizes, fragmentation may be called for.

The Internet has a rich variety of protocols related to the network layer. These include the data transport protocol, IP, but also the control protocols ICMP, ARP, and RARP, and the routing protocols OSPF and BGP. The Internet is rapidly running out of IP addresses, so a new version of IP, IPv6, has been developed.

Unlike the datagram-based Internet, ATM networks use virtual circuits inside. These must be set up before data can be transferred and torn down after transmission is completed. Quality of service and congestion control are major issues with ATM networks.

PROBLEMS

1. Give two example applications for which connection-oriented service is appropriate. Now give two examples for which connectionless service is best.

2. Are there any circumstances when a virtual circuit service will (or at least should) deliver packets out of order? Explain.

3. Datagram subnets route each packet as a separate unit, independent of all others. Virtual circuit subnets do not have to do this, since each data packet follows a predetermined route. Does this observation mean that virtual circuit subnets do not need the capability to route isolated packets from an arbitrary source to an arbitrary destination? Explain your answer.

4. Give three examples of protocol parameters that might be negotiated when a connection is set up.

5. Consider the following design problem concerning implementation of virtual circuit service. If virtual circuits are used internal to the subnet, each data packet must have a 3-byte header, and each router must tie up 8 bytes of storage for circuit identification. If datagrams are used internally, 15-byte headers are needed, but no router table space is required. Transmission capacity costs 1 cent per 10^6 bytes, per hop. Router memory can be purchased for 1 cent per byte and is depreciated over two years (business hours only). The statistically average session runs for 1000 sec, in which time 200 packets are transmitted. The mean packet requires four hops. Which implementation is cheaper, and by how much?

6. Assuming that all routers and hosts are working properly and that all software in both is free of all errors, is there any chance, however small, that a packet will be delivered to the wrong destination?

7. Give a simple heuristic for finding two paths through a network from a given source to a given destination that can survive the loss of any communication line (assuming two such paths exist). The routers are considered reliable enough, so it is not necessary to worry about the possibility of router crashes.

8. Consider the subnet of Fig. 5-15(a). Distance vector routing is used, and the following vectors have just come in to router *C*: from *B*: (5, 0, 8, 12, 6, 2); from *D*: (16, 12, 6, 0, 9, 10); and from *E*: (7, 6, 3, 9, 0, 4). The measured delays to *B*, *D*, and *E*, are 6, 3, and 5, respectively. What is *C*'s new routing table? Give both the outgoing line to use and the expected delay.

9. If delays are recorded as 8-bit numbers in a 50-router network, and delay vectors are exchanged twice a second, how much bandwidth per (full-duplex) line is chewed up by the distributed routing algorithm? Assume that each router has three lines to other routers.

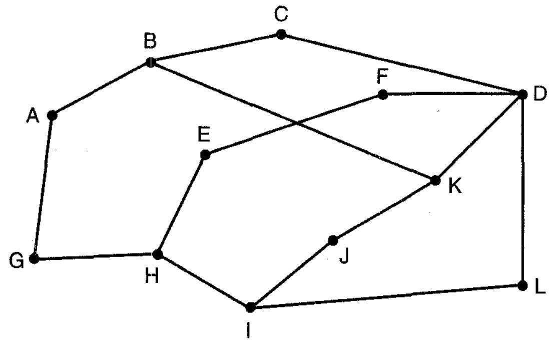
10. In Fig. 5-16 the Boolean OR of the two sets of ACF bits are 111 in every row. Is this just an accident here, or does it hold for all subnets under all circumstances?

11. For hierarchical routing with 4800 routers, what region and cluster sizes should be chosen to minimize the size of the routing table for a three-layer hierarchy?

12. In the text it was stated that when a mobile host is not at home, packets sent to its home LAN are intercepted by its home agent. For an IP network on an 802.3 LAN, how does the home agent accomplish this interception?

13. Looking at the subnet of Fig. 5-5, how many packets are generated by a broadcast from *B*, using
(a) reverse path forwarding?
(b) the sink tree?

14. Compute a multicast spanning tree for router *C* in the subnet below for a group with members at routers *A*, *B*, *C*, *D*, *E*, *F*, *I*, and *K*.



15. As a possible congestion control mechanism in a subnet using virtual circuits internally, a router could refrain from acknowledging a received packet until (1) it knows its last transmission along the virtual circuit was received successfully and (2) it has a free buffer. For simplicity, assume that the routers use a stop-and-wait protocol and that each virtual circuit has one buffer dedicated to it for each direction of traffic. If it

takes T sec to transmit a packet (data or acknowledgement) and there are n routers on the path, what is the rate at which packets are delivered to the destination host? Assume that transmission errors are rare, and that the host-router connection is infinitely fast.

16. A datagram subnet allows routers to drop packets whenever they need to. The probability of a router discarding a packet is p. Consider the case of a source host connected to the source router, which is connected to the destination router, and then to the destination host. If either of the routers discards a packet, the source host eventually times out and tries again. If both host-router and router-router lines are counted as hops, what is the mean number of
(a) hops a packet makes per transmission?
(b) transmissions a packet makes?
(c) hops required per received packet?

17. Give an argument why the leaky bucket algorithm should allow just one packet per tick, independent of how large the packet is.

18. The byte-counting variant of the leaky bucket algorithm is used in a particular system. The rule is that one 1024-byte packet, two 512-byte packets, etc. may be sent on each tick. Give a serious restriction of this system that was not mentioned in the text.

19. An ATM network uses a token bucket scheme for traffic shaping. A new token is put into the bucket every 5 μsec. What is the maximum sustainable net data rate (i.e., excluding header bits)?

20. A computer on a 6-Mbps network is regulated by a token bucket. The token bucket is filled at a rate of 1 Mbps. It is initially filled to capacity with 8 megabits. How long can the computer transmit at the full 6 Mbps?

21. Figure 5-27 shows four input characteristics for a proposed flow specification. Imagine that the maximum packet size is 1000 bytes, the token bucket rate is 10 million bytes/sec, the token bucket size is 1 million bytes, and the maximum transmission rate is 50 million bytes/sec. How long can a burst at maximum speed last?

22. A device accepts frames from the Ethernet to which it is attached. It removes the packet inside each frame, adds framing information around it, and transmits it over a leased telephone line (its only connection to the outside world) to an identical device at the other end. This device removes the framing, inserts the packet into a token ring frame, and transmits it to a local host over a token ring LAN. What would you call the device?

23. Is fragmentation needed in concatenated virtual circuit internets, or only in datagram systems?

24. Tunneling through a concatenated virtual circuit subnet is straightforward: the multiprotocol router at one end just sets up a virtual circuit to the other end and passes packets through it. Can tunneling also be used in datagram subnets? If so, how?

25. An IP datagram using the *Strict source routing* option has to be fragmented. Do you think the option is copied into each fragment, or is it sufficient to just put it in the first fragment? Explain your answer.

26. Suppose that instead of using 16 bits for the network part of a class B address, 20 bits had been used. How many class B networks would there have been?

27. Convert the IP address whose hexadecimal representation is C22F1582 to dotted decimal notation.

28. A class B network on the Internet has a subnet mask of 255.255.240.0. What is the maximum number of hosts per subnet?

29. You have just explained the ARP protocol to a friend. When you are all done, he says: "I've got it. ARP provides a service to the network layer, so it is part of the data link layer." What do you say to him?

30. ARP and RARP both map addresses from one space to another. In this respect, they are similar. However, their implementations are fundamentally different. In what major way do they differ?

31. Describe a way to do reassembly of IP fragments at the destination.

32. Most IP datagram reassembly algorithms have a timer to avoid having a lost fragment tie up reassembly buffers forever. Suppose a datagram is fragmented into four fragments. The first three fragments arrrive, but the last one is delayed. Eventually the timer goes off and the three fragments in the receiver's memory are discarded. A little later, the last fragment stumbles in. What should be done with it?

33. Most IP routing protocols use number of hops as the metric to be minimized when doing routing computations. For ATM networks, number of hops is not terribly important. Why not? *Hint*: Take a look at Chap. 2. to see how ATM switches work. Do they use store-and-forward?

34. In both IP and ATM, the checksum covers only the header and not the data. Why do you suppose this design was chosen?

35. A person who lives in Boston travels to Minneapolis, taking her portable computer with her. To her surprise, the LAN at her destination in Minneapolis is a wireless IP LAN, so she does not have to plug in. Is it still necessary to go through the entire business with home agents and foreign agents to make email and other traffic arrive correctly?

36. IPv6 uses 16-byte addresses. If a block of 1 million addresses is allocated every picosecond, how long will the addresses last?

37. The *Protocol* field used in the IPv4 header is not present in the fixed IPv6 header. Why not?

38. When the IPv6 protocol is introduced, does the ARP protocol have to be changed? If so, are the changes conceptual or technical?

39. In Chap. 1, we classified interactions between the network and the hosts using four classes of primitives: *request*, *indication*, *response*, and *confirm*. Classify the SETUP and CONNECT messages of Fig. 5-65 into these categories.

40. A new virtual circuit is being set up in an ATM network. Between the source and destination hosts lie three ATM switches. How many messages (including acknowledgements) will be sent to establish the circuit?

41. The logic used to construct the table of Fig. 5-67 is simple: the lowest unused *VPI* is always assigned to a connection. If a new virtual circuit is requested between NY and Denver, which *VPI* will be assigned to it?

42. In Fig. 5-73(c), if a cell arrives early, the next one is still due at $t_1 + 2T$. Suppose that the rule were different, namely that the next cell was expected at $t_2 + T$, and the sender made maximum use of this rule. What maximum peak cell rate could then be achieved? For $T = 10$ μsec and $L = 2$ μsec, give the original and new peak cell rates, respectively.

43. What is the maximum burst length on an 155.52 Mbps ATM ABR connection whose *PCR* value is 200,000 and whose *L* value is 25 μsec?

44. Write a program to simulate routing using flooding. Each packet should contain a counter that is decremented on each hop. When the counter gets to zero, the packet is discarded. Time is discrete, with each line handling one packet per time interval. Make three versions of the program: all lines are flooded, all lines except the input line are flooded, and only the (statically chosen) best *k* lines are flooded. Compare flooding with deterministic routing ($k = 1$) in terms of delay and bandwidth used.

45. Write a program that simulates a computer network using discrete time. The first packet on each router queue makes one hop per time interval. Each router has only a finite number of buffers. If a packet arrives and there is no room for it, it is discarded and not retransmitted. Instead, there is an end-to-end protocol, complete with timeouts and acknowledgement packets, that eventually regenerates the packet from the source router. Plot the throughput of the network as a function of the end-to-end timeout interval, parametrized by error rate.