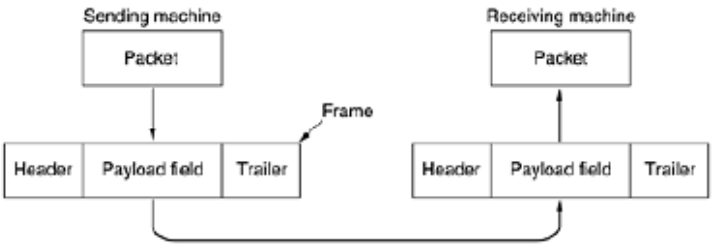
Unit 3
Data Link Layer

Data Link Layer Design Issues

The data link layer has a number of specific functions it can carry out. These functions include

1. Providing a well-defined service interface to the network layer.
2. Dealing with transmission errors.
3. Regulating the flow of data so that slow receivers are not swamped by fast senders.

To accomplish these goals, the data link layer takes the packets it gets from the network layer and encapsulates them into frames for transmission. Each frame contains a frame header, a payload field for holding the packet, and a frame trailer, as illustrated in Figure. Frame management forms the heart of what the data link layer does. In the following sections we will examine all the above-mentioned issues in detail.



Error Control

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

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

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

This possibility is dealt with by introducing timers into the data link layer. When the sender transmits a frame, it generally also starts a timer. The timer is set to expire after an interval long enough for the frame

to reach the destination, be processed there, and have the acknowledgement propagate back to the sender. Normally, the frame will be correctly received and the acknowledgement will get back before the timer runs out, in which case the timer will be canceled.

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

Error Detection and Correction

The telephone system has three parts: the switches, the interoffice trunks, and the local loops. The first two are now almost entirely digital in most developed countries. The local loops are still analog twisted copper pairs and will continue to be so for years due to the enormous expense of replacing them. While errors are rare on the digital part, they are still common on the local loops. Furthermore, wireless communication is becoming more common, and the error rates here are orders of magnitude worse than on the interoffice fiber trunks. The conclusion is: transmission errors are going to be with us for many years to come. We have to learn how to deal with them.

As a result of the physical processes that generate them, errors on some media (e.g., radio) tend to come in bursts rather than singly. Having the errors come in bursts has both advantages and disadvantages over isolated single-bit errors. On the advantage side, computer data are always sent in blocks of bits. Suppose that the block size is 1000 bits and the error rate is 0.001 per bit. If errors were independent, most blocks would contain an error. If the errors came in bursts of 100 however, only one or two blocks in 100 would be affected, on average. The disadvantage of burst errors is that they are much harder to correct than are isolated errors.

Error-Correcting Codes

Network designers have developed two basic strategies for dealing with errors. One way is to include enough redundant information along with each block of data sent, to enable the receiver to deduce what the transmitted data must have been. The other way is to include only enough redundancy to allow the receiver to deduce that an error occurred, but not which error, and have it request a retransmission. The former strategy uses error-correcting codes and the latter uses error-detecting codes. The use of error-correcting codes is often referred to as forward error correction.

Each of these techniques occupies a different ecological niche. On channels that are highly reliable, such as fiber, it is cheaper to use an error detecting code and just retransmit the occasional block found to be faulty. However, on channels such as wireless links that make many errors, it is better to add enough redundancy to each block for the receiver to be able to figure out what the original block was, rather than relying on a retransmission, which itself may be in error.

To understand how errors can be handled, it is necessary to look closely at what an error really is. Normally, a frame consists of m data (i.e., message) bits and r redundant, or check, bits. Let the total length be n (i.e., n = m + r). An n-bit unit containing data and check bits is often referred to as an n-bit codeword.

Given any two code words, say, 10001001 and 10110001, it is possible to determine how many corresponding bits differ. In this case, 3 bits differ. To determine how many bits differ, just exclusive OR the two code words and count the number of 1 bits in the result, for example:

```
10001001
10110001
00111000
```

Framing

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

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

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

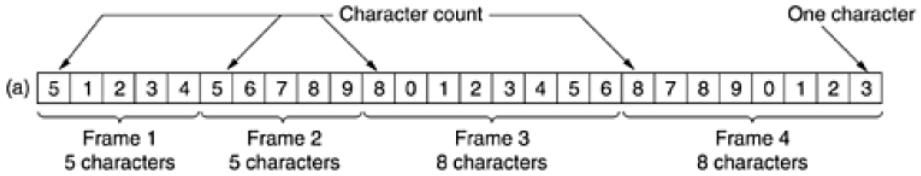
Since it is too risky to count on timing to mark the start and end of each frame, other methods have been devised. In this section we will look at four methods:

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

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

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

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



Error-Detecting Codes

Error-correcting codes are widely used on wireless links, which are notoriously noisy and error prone when compared to copper wire or optical fibers. Without error-correcting codes, it would be hard to get

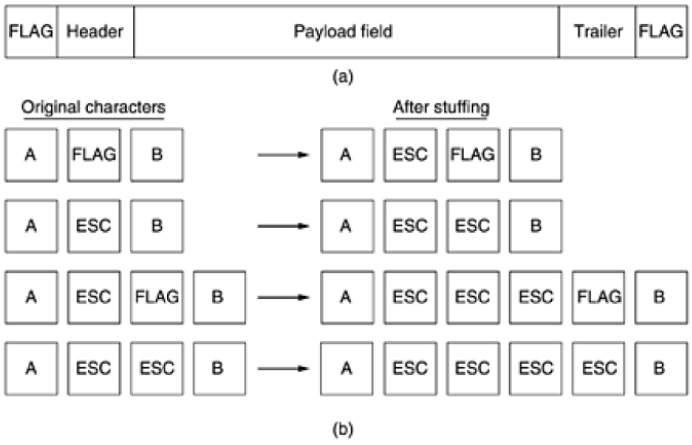
anything through. However, over copper wire or fiber, the error rate is much lower, so error detection and retransmission is usually more efficient there for dealing with the occasional error.

As a simple example, consider a channel on which errors are isolated and the error rate is 10^{-6} per bit. Let the block size be 1000 bits. To provide error correction for 1000-bit blocks, 10 check bits are needed; a megabit of data would require 10,000 check bits. To merely detect a block with a single 1-bit error, one parity bit per block will suffice. Once every 1000 blocks, an extra block (1001 bits) will have to be transmitted. The total overhead for the error detection + retransmission method is only 2001 bits per megabit of data, versus 10,000 bits for a Hamming code.

If a single parity bit is added to a block and the block is badly garbled by a long burst error, the probability that the error will be detected is only 0.5, which is hardly acceptable. The odds can be improved considerably if each block to be sent is regarded as a rectangular matrix n bits wide and k bits high, as described above. A parity bit is computed separately for each column and affixed to the matrix as the last row. The matrix is then transmitted one row at a time. When the block arrives, the receiver checks all the parity bits. If any one of them is wrong, the receiver requests a retransmission of the block. Additional retransmissions are requested as needed until an entire block is received without any parity errors.

```
  10011011      00110011      11110000      01010101
+ 11001010    + 11001101    - 10100110    - 10101111
----------    ----------    ----------    ----------
  01010001      11111110      01010110      11111010
```

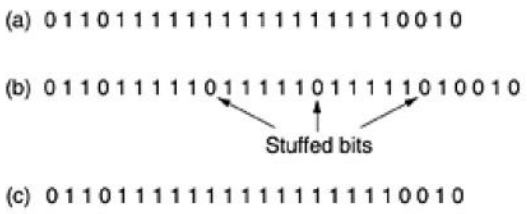
The second framing method gets around the problem of resynchronization after an error by having each frame start and end with special bytes. In the past, the starting and ending bytes were different, but in recent years most protocols have used the same byte, called a flag byte, as both the starting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



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

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit. Just as byte stuffing is completely transparent to the network layer in both

computers, so is bit stuffing. If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver's memory as 01111110.



Error Control

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

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

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

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

Error Detection and Correction

Error-Correcting Codes

Network designers have developed two basic strategies for dealing with errors. One way is to include enough redundant information along with each block of data sent, to enable the receiver to deduce what the transmitted data must have been. The other way is to include only enough redundancy to allow the receiver to deduce that an error occurred, but not which error, and have it request a retransmission. The former strategy uses error-correcting codes and the latter uses error-detecting codes. The use of error-correcting codes is often referred to as forward error correction.

Each of these techniques occupies a different ecological niche. On channels that are highly reliable, such as fiber, it is cheaper to use an error detecting code and just retransmit the occasional block found to be faulty. However, on channels such as wireless links that make many errors, it is better to add enough

redundancy to each block for the receiver to be able to figure out what the original block was, rather than relying on a retransmission, which itself may be in error.

To understand how errors can be handled, it is necessary to look closely at what an error really is. Normally, a frame consists of m data (i.e., message) bits and r redundant, or check, bits. Let the total length be n (i.e., n = m + r). An n-bit unit containing data and check bits is often referred to as an n-bit codeword.

Hamming Code:

In most data transmission applications, all 2^m possible data messages are legal, but due to the way the check bits are computed, not all of the 2^n possible codewords are used. Given the algorithm for computing the check bits, it is possible to construct a complete list of the legal codewords, and from this list find the two codewords whose Hamming distance is minimum. This distance is the Hamming distance of the complete code.

The error-detecting and error-correcting properties of a code depend on its Hamming distance. To detect d errors, you need a distance d + 1 code because with such a code there is no way that d single-bit errors can change a valid codeword into another valid codeword. When the receiver sees an invalid codeword, it can tell that a transmission error has occurred. Similarly, to correct d errors, you need a distance 2d + 1 code because that way the legal codewords are so far apart that even with d changes, the original codeword is still closer than any other codeword, so it can be uniquely determined.

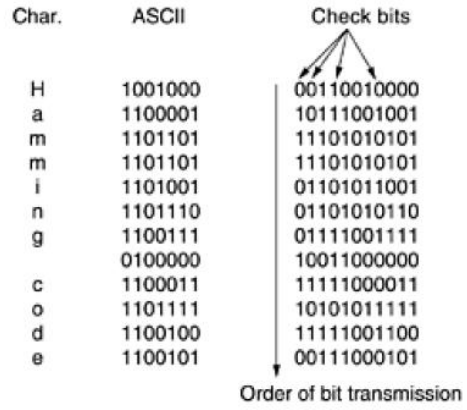
As a simple example of an error-detecting code, consider a code in which a single parity bit is appended to the data. The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd). For example, when 1011010 is sent in even parity, a bit is added to the end to make it 10110100. With odd parity 1011010 becomes 10110101. A code with a single parity bit has a distance 2, since any single-bit error produces a codeword with the wrong parity. It can be used to detect single errors.

As a simple example of an error-correcting code, consider a code with only four valid codewords:

0000000000, 0000011111, 1111100000, and 1111111111

This code has a distance 5, which means that it can correct double errors. If the codeword 0000000111 arrives, the receiver knows that the original must have been 0000011111. If, however, a triple error changes 0000000000 into 0000000111, the error will not be corrected properly.

Imagine that we want to design a code with m message bits and r check bits that will allow all single errors to be corrected. Each of the 2^m legal messages has n illegal codewords at a distance 1 from it. These are formed by systematically inverting each of the n bits in the n-bit codeword formed from it. Thus, each of the 2^m legal messages requires n + 1 bit patterns dedicated to it. Since the total number of bit patterns is 2^n, we must have $(n + 1)2^m 2^n$. Using n = m + r, this requirement becomes $(m + r + 1)\ 2^r$. Given m, this puts a lower limit on the number of check bits needed to correct single errors.

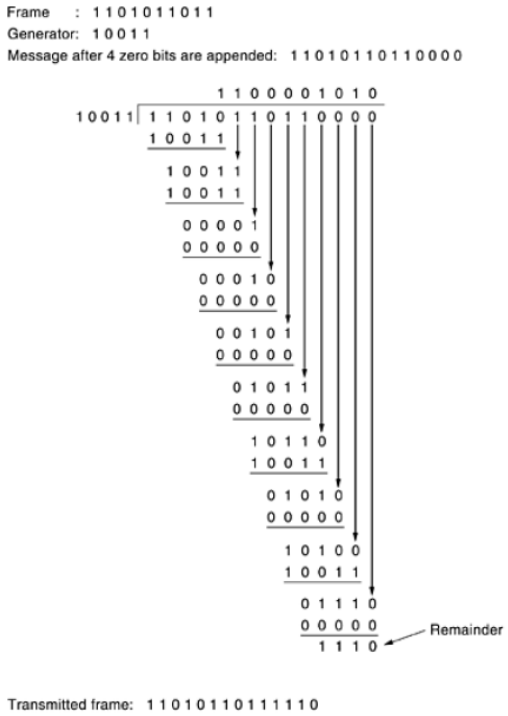


Error-Detecting Codes

Error-correcting codes are widely used on wireless links, which are notoriously noisy and error prone when compared to copper wire or optical fibers. Without error-correcting codes, it would be hard to get anything through. However, over copper wire or fiber, the error rate is much lower, so error detection and retransmission is usually more efficient there for dealing with the occasional error.

As a simple example, consider a channel on which errors are isolated and the error rate is 10^{-6} per bit. Let the block size be 1000 bits. To provide error correction for 1000-bit blocks, 10 check bits are needed; a megabit of data would require 10,000 check bits. To merely detect a block with a single 1-bit error, one parity bit per block will suffice. Once every 1000 blocks, an extra block (1001 bits) will have to be transmitted. The total overhead for the error detection + retransmission method is only 2001 bits per megabit of data, versus 10,000 bits for a Hamming code

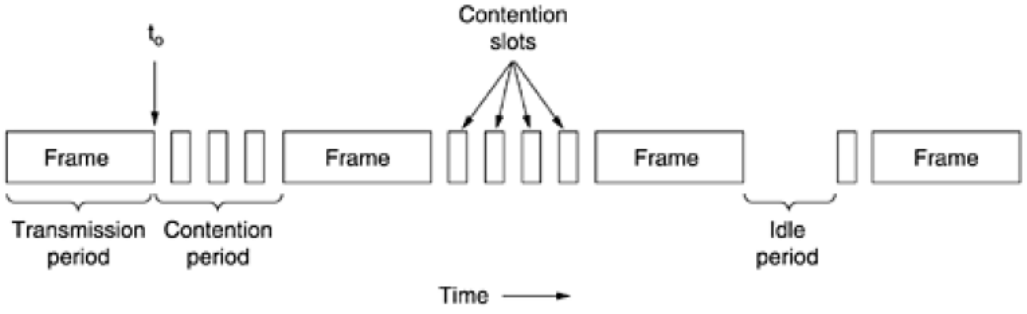
The algorithm for computing the checksum is as follows:
1. Let r be the degree of G(x). Append r zero bits to the low-order end of the frame so it now contains m + r bits and corresponds to the polynomial xMr(x).
2. Divide the bit string corresponding to G(x) into the bit string corresponding to xMr(x), using modulo 2 division.
3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to xMr(x) using modulo 2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial T(x).



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** (**CSMA with Collision Detection**) is widely used on LANs in the MAC sublayer. In particular, it is the basis of the popular Ethernet LAN, so it is worth devoting some time to looking at it in detail.

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.



Collision-Free Protocols

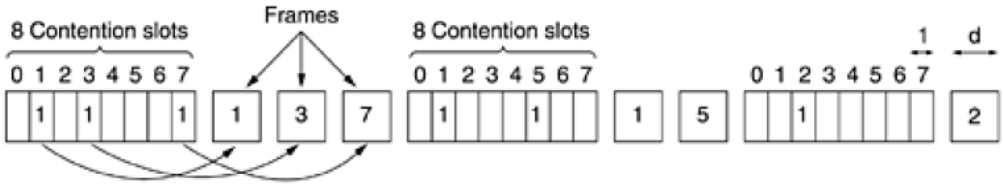
Although collisions do not occur with CSMA/CD once a station has unambiguously captured the channel, they can still occur during the contention period. These collisions adversely affect the system performance, especially when the cable is long (i.e., large τ) and the frames are short. And CSMA/CD is not universally applicable. 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. Most of these are not currently used in major systems, but in a rapidly changing field, having some protocols with excellent properties available for future systems is often a good thing.

In the protocols to be described, we assume that there are exactly *N* stations, each with a unique address from 0 to *N* - 1 "wired" into it. It does not matter that some stations may be inactive part of the time. We also assume that propagation delay is negligible. 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 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).



Data Link Layer Switching

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. Bridges examine the data layer link addresses to do routing. Since they are not supposed to examine the payload field of the frames they route, they can transport IPv4 (used in the Internet now), IPv6 (will be used in the Internet in the future), AppleTalk, ATM, OSI, or any other kinds of packets. In contrast, *routers* examine the addresses in packets and route based on them. Although this seems like a clear division between bridges and routers, some modern developments, such as the advent of switched Ethernet, have muddied the waters, as we will see later. In the following sections we will look at bridges and switches, especially for connecting different 802 LANs. For a comprehensive treatment of bridges, switches, and related topics, see (Perlman, 2000).

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 laser links than to run a single cable over the entire site.

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. 4-39. 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.