7 Interconnection Networks

"The Medium is the Message" because it is the medium that shapes and controls the search and form of human associations and actions.

Marshall McLuhan
Understanding Media (1964)

The marvels—of film, radio, and television—are marvels of one-way communication, which is not communication at all.

Milton Mayer
On the Remote Possibility of Communication (1967)

7.1 Introduction

Thus far we have covered the components of a single computer, which has been the traditional focus of computer architecture. In this chapter we see how to connect computers together, forming a community of computers. Figure 7.1 shows the generic components of this community: computer nodes, hardware and software interfaces, links to the interconnection network, and the interconnection network. Interconnection networks are also called *networks* or *communication subnets*, and nodes are sometimes called *end systems* or *hosts*. This topic is vast, with whole books written about portions of this figure. The goal of this chapter is to help you understand the architectural implications of interconnection network technology, providing introductory explanations of the key ideas and references to more detailed descriptions.

Let's start with the generic types of interconnections. Depending on the number of nodes and their proximity, these interconnections are given different names:

- *Massively parallel processor (MPP) network*—This interconnection network can connect thousands of nodes, and the maximum distance is typically less than 25 meters. The nodes are typically found in a row of adjacent cabinets.

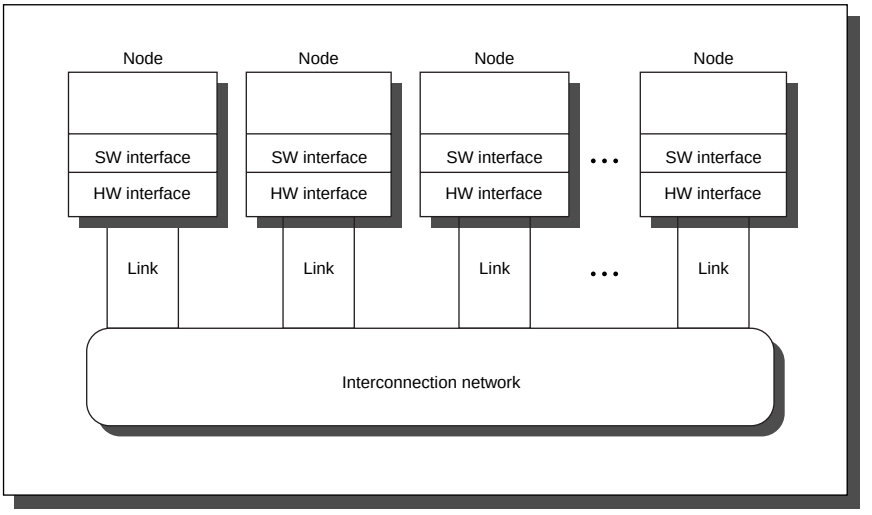

FIGURE 7.1 Drawing of the generic interconnection network.

- *Local area network (LAN)*—This device connects hundreds of computers, and the distance is up to a few kilometers. Unlike the MPP network, the LAN connects computers distributed throughout a building. The traffic is mostly many-to-one, such as between clients and server, while MPP traffic is often between all nodes.
- *Wide area network (WAN)*—Also called *long haul network*, the WAN connects computers distributed throughout the world. WANs include thousands of computers, and the maximum distance is thousands of kilometers.

The connection of two or more interconnection networks is called *internetworking*, which relies on software standards to convert information from one kind of network to another.

These three types of interconnection networks have been designed and sustained by three different cultures—the MPP, workstation, and telecommunications communities—each using its own dialects and its own favorite approaches to the goal of interconnecting autonomous computers.

This chapter gives a common framework for evaluating all interconnection networks, using a single set of terms to describe the basic alternatives. Figure 7.21 in section 7.7 gives several other examples of each of these interconnection networks. As we shall see, some components are common to each type and some are quite different.

We begin the chapter by exploring the design and performance of a simple network to introduce the ideas. We then consider the following problems: where

to attach the interconnection network, which media to use as the interconnect, how to connect many computers together, and what are the practical issues for commercial networks. We follow with examples illustrating the trade-offs for each type of network, explore internetworking, and conclude with the traditional ending of the chapters in this book.

7.2 A Simple Network

To explain the complexities and concepts of networks, this section describes a simple network of two computers. We then describe the software steps for these two machines to communicate. The remainder of the section gives a detailed and then a simple performance model, including several examples to see the implications of key network parameters.

Suppose we want to connect two computers together. Figure 7.2 shows a simple model with a unidirectional wire from machine A to machine B and vice versa. At the end of each wire is a first-in-first-out (FIFO) queue to hold the data. In this simple example each machine wants to read a word from the other's memory. The information sent between machines over an interconnection network is called a *message.*

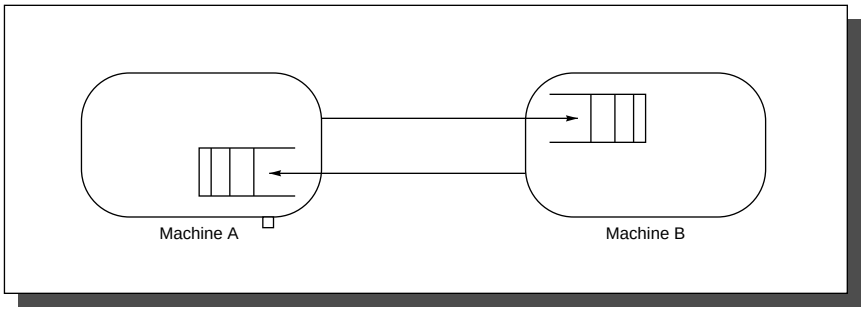

FIGURE 7.2 A simple network connecting two machines.

For one machine to get data from the other, it must first send a request containing the address of the data it desires from the other node. When a request arrives, the machine must send a reply with the data. Hence each message must have at least 1 bit in addition to the data to determine whether the message is a new request or a reply to an earlier request. The network must distinguish between information needed to deliver the message, typically called the *header* or the *trailer* depending on where it is relative to the data, and the *payload*, which contains the data. Figure 7.3 shows the format of messages in our simple network. This example shows a single-word payload, but messages in some interconnection networks can include hundreds of words.

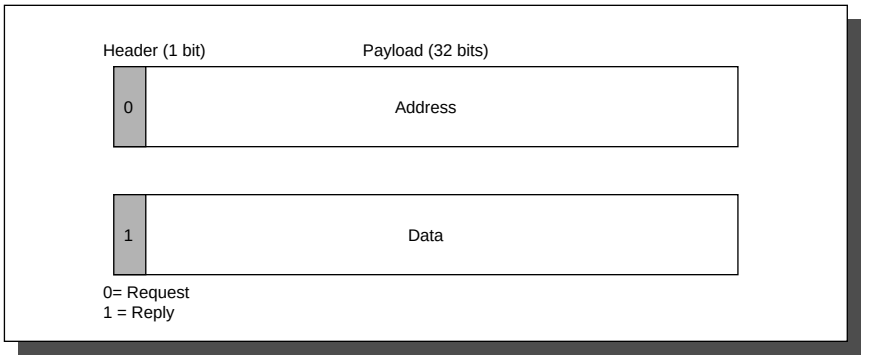

FIGURE 7.3 Message format for our simple network. Messages must have extra information beyond the data.

All interconnection networks involve software. Even this simple example invokes software to translate requests and replies into messages with the appropriate headers. An application program must usually cooperate with the operating system to send a message to another machine, since the network will be shared with all the processes running on the two machines, and the operating system cannot allow messages for one process to be received by another. Thus the messaging software must have some way to distinguish between processes; this distinction may be included in an expanded header. Although hardware support can reduce the amount of work, most is done by software.

In addition to protection, network software is often responsible for ensuring that messages are reliably delivered. The twin responsibilities are ensuring that the message is not garbled in transit, or lost in transit.

The first responsibility is met by adding a *checksum* field to the message format; this redundant information is calculated when the message is first sent and checked upon receipt. The receiver then sends an acknowledgment if the message passes the test.

One way to meet the second responsibility is to have a timer record the time each message is sent and to presume the message is lost if the timer expires before an acknowledgment arrives. The message is then re-sent.

The software steps to send a message are as follows:

1. The application copies data to be sent into an operating system buffer.
2. The operating system calculates the checksum, includes it in the header or trailer of the message, and then starts the timer.
3. The operating system sends the data to the network interface hardware and tells the hardware to send the message.

Message reception is in just the reverse order:

3. The system copies the data from the network interface hardware into the operating system buffer.
2. The system calculates the checksum over the data. If the checksum matches the sender's checksum, the receiver sends an acknowledgment back to the sender; if not, it deletes the message, assuming that the sender will resend the message when the associated timer expires.
1. If the data pass the test, the system copies the data to the user's address space and signals the application to continue.

The sender must still react to the acknowledgment:

- When the sender gets the acknowledgment, it releases the copy of the message from the system buffer.
- If the sender gets the time-out instead, it resends the data and restarts the timer.

Here we assume that the operating system keeps the message in its buffer to support retransmission in case of failure. Figure 7.4 shows how the message format looks now.

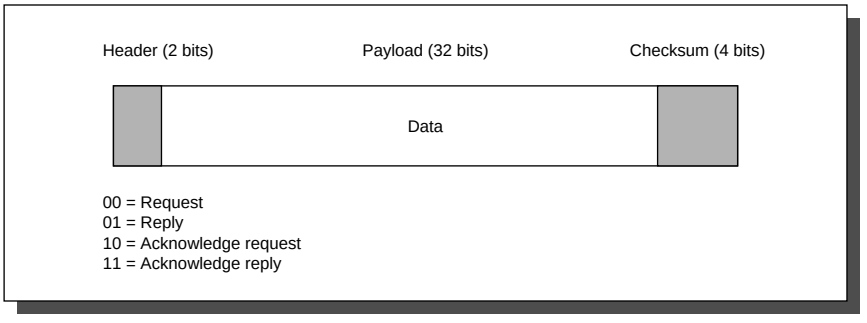

FIGURE 7.4 Message format for our simple network. Note that the checksum is in the trailer.

The sequence of steps that software follows to communicate is called a *protocol* and generally has the symmetric but reversed steps between sending and receiving. Our example is similar to the *UDP/IP* protocol used by some UNIX systems. Note that this protocol is for sending a *single* message. When an application does not require a response before sending the next message, the sender can overlap the time to send with the transmission delays and the time to receive.

A protocol must handle many more issues than reliability. For example, if two machines are from different manufacturers, they might order bytes differently

within a word (see section 2.3 of Chapter 2). The software must reverse the order of bytes in each word as part of the delivery system. It must also guard against the possibility of duplicate messages if a delayed message were to become unstuck. Finally, it must work when the receiver's FIFO becomes full, suggesting feedback to control the flow of messages from the sender (see section 7.5).

Now that we have covered the steps in sending and receiving a message, we can discuss performance. Figure 7.5 shows the many performance parameters of interconnection networks. These terms are often used loosely, leading to confusion, so we define them here precisely:

- *Bandwidth*—This most widely used term refers to the maximum rate at which the interconnection network can propagate information once the message enters the network. Traditionally, the headers and trailers as well as the payload are counted in the bandwidth calculation, and the units are megabits/second rather than megabytes/second. The term *throughput* is sometimes used to mean network bandwidth delivered to an application.
- *Time of flight*—The time for the first bit of the message to arrive at the receiver, including the delays due to repeaters or other hardware in the network. Time of flight can be milliseconds for a WAN or nanoseconds for an MPP.
- *Transmission time*—The time for the message to pass through the network (not including time of flight) and equal to the size of the message divided by the bandwidth. This measure assumes there are no other messages to contend for the network.

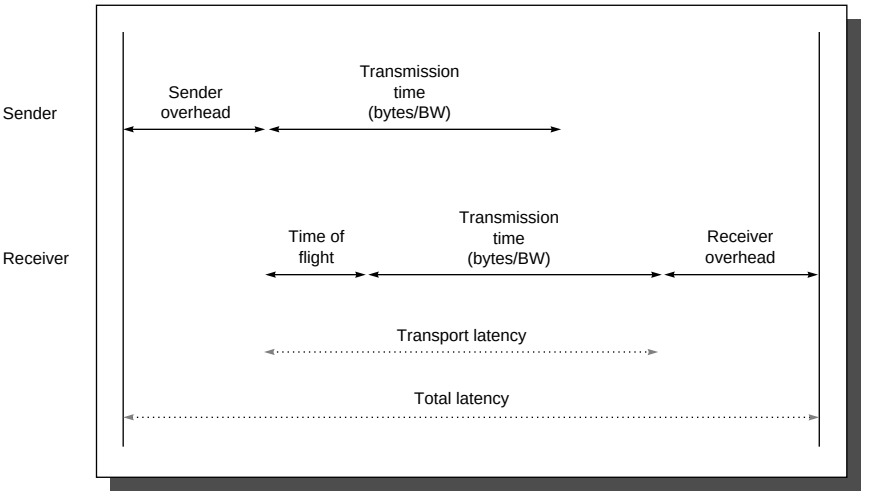

FIGURE 7.5 Performance parameters of interconnection networks. Depending on whether it is an MPP, LAN, or WAN, the relative lengths of the time of flight and transmission may be quite different from those shown here. (Based on a presentation by Greg Papadopolous, Sun Microsystems.)

- *Transport latency*—The sum of time of flight and transmission time, it is the time that the message spends in the interconnection network, not including the overhead of injecting the message into the network nor pulling it out when it arrives.
- *Sender overhead*—The time for the processor to inject the message into the interconnection network, including both hardware and software components. Note that the processor is busy for the entire time, hence the use of the term *overhead*. Once the processor is free, any subsequent delays are considered part of the transport latency.
- *Receiver overhead*—The time for the processor to pull the message from the interconnection network, including both hardware and software components. In general, the receiver overhead is larger than the sender overhead: for example, the receiver may pay the cost of an interrupt.

The total latency of a message can be expressed algebraically:

$$\text{Total latency} = \text{Sender overhead} + \text{Time of flight} + \frac{\text{Message size}}{\text{Bandwidth}} + \text{Receiver overhead}$$

As we shall see, for many applications and networks, the overheads dominate the total message latency.

EXAMPLE Assume a network with a bandwidth of 10 Mbits/second has a sending overhead of 230 microseconds and a receiving overhead of 270 microseconds. Assume two machines are 100 meters apart and one wants to send a 1000-byte message to another (including the header), and the message format allows 1000 bytes in a single message. Calculate the total latency to send the message from one machine to another. Next, perform the same calculation but assume the machines are now 1000 km apart.

ANSWER The speed of light is 299,792.5 kilometers per second, and signals propagate at about 50% of the speed of light in a conductor, so time of flight can be estimated. Let's plug the parameters for the shorter distance into the formula above:

$$\begin{aligned}\text{Total latency} &= \text{Sender overhead} + \text{Time of flight} + \frac{\text{Message size}}{\text{Bandwidth}} + \text{Receiver overhead}\\ &= 230\ \mu\text{secs} + \frac{0.1\text{km}}{0.5 \times 299{,}792.5\ \text{km/sec}} + \frac{1000\ \text{bytes}}{10\ \text{Mbits/sec}} + 270\ \mu\text{secs}\\ &= 230\ \mu\text{secs} + \frac{0.1 \times 10^6}{0.5 \times 299{,}792.5}\ \mu\text{secs} + \frac{1000 \times 8}{10}\ \mu\text{secs} + 270\ \mu\text{secs}\\ &= 230\ \mu\text{secs} + 0.67\ \mu\text{secs} + 800\ \mu\text{secs} + 270\ \mu\text{secs}\\ &= 1301\mu\text{secs}\end{aligned}$$

Substituting the longer distance into the third equation yields

$$\begin{aligned}\text{Total latency} &= 230\ \mu\text{secs} + \frac{1000 \times 10^6}{0.5 \times 299{,}792.5}\ \mu\text{secs} + \frac{1000 \times 8}{10}\ \mu\text{secs} + 270\ \mu\text{secs}\\ &= 230\ \mu\text{secs} + 6671\mu\text{secs} + 800\ \mu\text{secs} + 270\ \mu\text{secs}\\ &= 7971\ \mu\text{secs}\end{aligned}$$

The increased fraction of the latency required by time of flight for long distances, as well as the greater likelihood of errors over long distances, are why wide area networks use more sophisticated and time-consuming protocols. Increased latency affects the structure of programs that try to hide this latency, requiring quite different solutions if the latency is 1, 100, or 10,000 microseconds.

As mentioned above, when an application does not require a response before sending the next message, the sender can overlap the sending overhead with the transport latency and receiver overhead. ■

We can simplify the performance equation by combining sender overhead, receiver overhead, and time of flight into a single term called *Overhead*:

$$\text{Total latency} \approx \text{Overhead} + \frac{\text{Message size}}{\text{Bandwidth}}$$

We can use this formula to calculate the effective bandwidth delivered by the network as message size varies:

$$\text{Effective bandwidth} = \frac{\text{Message size}}{\text{Total latency}}$$

Let's use this simpler equation to explore the impact of overhead and message size on effective bandwidth.

EXAMPLE Plot the effective bandwidth versus message size for overheads of 1, 25, and 500 microseconds and for network bandwidths of 10, 100, and 1000 Mbits/second. Vary message size from 16 bytes to 4 megabytes. For what message sizes is the effective bandwidth virtually the same as the raw network bandwidth? Assuming a 500-microsecond overhead, for what message sizes is the effective bandwidth always less than 10 Mbits/second?

ANSWER Figure 7.6 plots effective bandwidth versus message size using the simplified equation above. The notation "oX,bwY" means an overhead of X microseconds and a network bandwidth of Y Mbits/second. Message sizes must be four megabytes for effective bandwidth to be about the same as network bandwidth, thereby amortizing the cost of high overhead. Assuming the high overhead, message sizes less than 4096 bytes will not break the 10 Mbits/second barrier no matter what the actual network bandwidth.

Thus we must lower overhead as well as increase network bandwidth unless messages are very large. ■

Many applications send far more small messages than large messages. Figure 7.7 shows the size of Network File System (NFS) messages for 239 machines at Berkeley collected over a period of one week. One plot is cumulative in messages sent, and the other is cumulative in data bytes sent. The maximum NFS message size is just over 8 KB, yet 95% of the messages are less than 192 bytes long.

Even this simple network has brought up the issues of protection, reliability, heterogeneity, software protocols, and a more sophisticated performance model. The next four sections address other key questions:

- Where do you connect the network to the computer?
- Which media are available to connect computers together?
- What issues arise if you want to connect more than two computers?
- What practical issues arise for commercial networks?

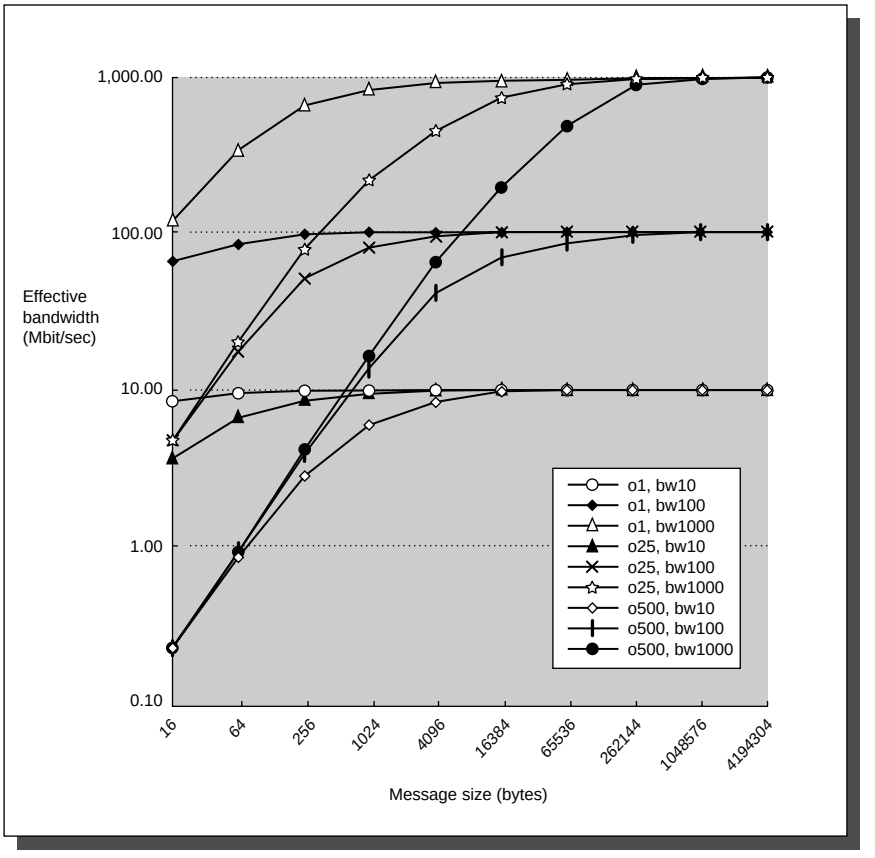

FIGURE 7.6 Bandwidth delivered versus message size for overheads of 1, 25, and 500 microseconds and for network bandwidths of 10, 100, and 1000 Mbits/second. The notation "oX,bwY" means an overhead of X microseconds and a network bandwidth of Y Mbits/second. Note that with 500 microseconds of overhead and a network bandwidth of 1000 Mbits/second, only the 4-MB message size gets an effective bandwidth of 1000 Mbits/second. In fact, message sizes must be greater than 4 KB for the effective bandwidth to exceed 10 Mbits/second.

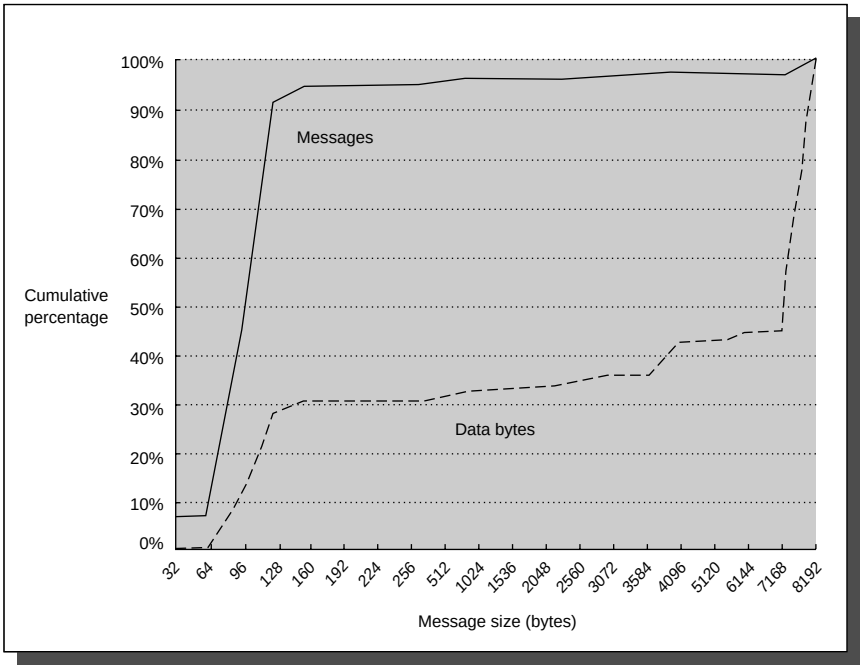

FIGURE 7.7 Cumulative percentage of messages and data transferred as message size varies for NFS traffic in the Computer Science Department at University of California at Berkeley. Each x-axis entry includes all bytes up to the next one; e.g., 32 represents 32 bytes to 63 bytes. More than half the bytes are sent in 8-KB messages, but 95% of the messages are less than 192 bytes. Figure 7.39 (page 622) shows the details of this measurement.

7.3 Connecting the Interconnection Network to the Computer

Where the network attaches to the computer affects both the network interface hardware and software. Questions include whether to use the memory bus or the I/O bus, whether to use polling or interrupts, and how to avoid invoking the operating system.

Computers have a hierarchy of buses with different cost/performance. For example, a personal computer in 1995 has a memory bus, a PCI bus for fast I/O devices, and an ISA bus for slow I/O devices. I/O buses follow open standards and have less stringent electrical requirements. Memory buses, on the other hand, provide higher bandwidth and lower latency than I/O buses. Typically, MPPs plug into the memory bus, and LANs and WANs plug into the I/O bus.

Where to connect the network to the machine depends on the performance goals and whether you hope to buy a standard network interface card or are willing to design or buy one that only works with the memory bus on your model of computer.

The location of the network connection significantly affects the software interface to the network as well as the hardware. As mentioned in section 6.6, one key is whether the interface is consistent with the processor's caches: the sender may have to flush the cache before each send, and the receiver may have to flush its cache before each receive to prevent the stale data problem. Such flushes increase send and receive overhead. A memory bus is more likely to be cache-coherent than an I/O bus and therefore more likely to avoid these extra cache flushes.

A related question of where to connect to the computer is how to connect to the software: Do you use programmed I/O or direct memory access (DMA) to send a message? (See section 6.6.) In general, large messages are best sent by DMA. Whether to use DMA to send small messages depends on the efficiency of the interface to the DMA. The DMA interface is usually memory-mapped, and so each interaction is typically at the speed of main memory rather than of a cache access. If DMA setup takes many accesses, each running at uncached memory speeds, then the sender overhead may be so high that it is faster to simply send the data directly to the interface.

Interconnection networks follow biblical advice: It's easier to send than to receive. One question is how the receiver should be notified when a message arrives. Should it poll the network interface waiting for a message to arrive, or should it perform other tasks and then pay the overhead to service an interrupt when it arrives? The issue is the time wasted polling before the message arrives versus the time wasted in interrupting the processor and restoring its state.

EXAMPLE The CM-5 is an MPP that allows users to send messages without invoking the operating system and allows the receiver to either poll or use interrupts. First plot the average overhead for polling and interrupts as a function of message arrival. Then propose a message reception scheme for the CM-5 that will work well as the rate varies. The time per poll is 1.6 microseconds: 0.6 to poll the interface card and 1.0 to check the type of message and get it from the interface card. The time per interrupt is 19 microseconds. The times are 4.9 microseconds and 3.75 microseconds to enable or disable interrupts, respectively, because the CM-5 operating system kernel must be invoked.

ANSWER For polling, the wasted time is simply the time between messages less the time to execute the simplest code to handle the message, which takes 0.5 microseconds. Interrupts cannot process messages any faster than the interrupt overhead time plus the time to handle a message, so the fastest time between interrupts is 19.5 microseconds. Figure 7.8 plots these curves.

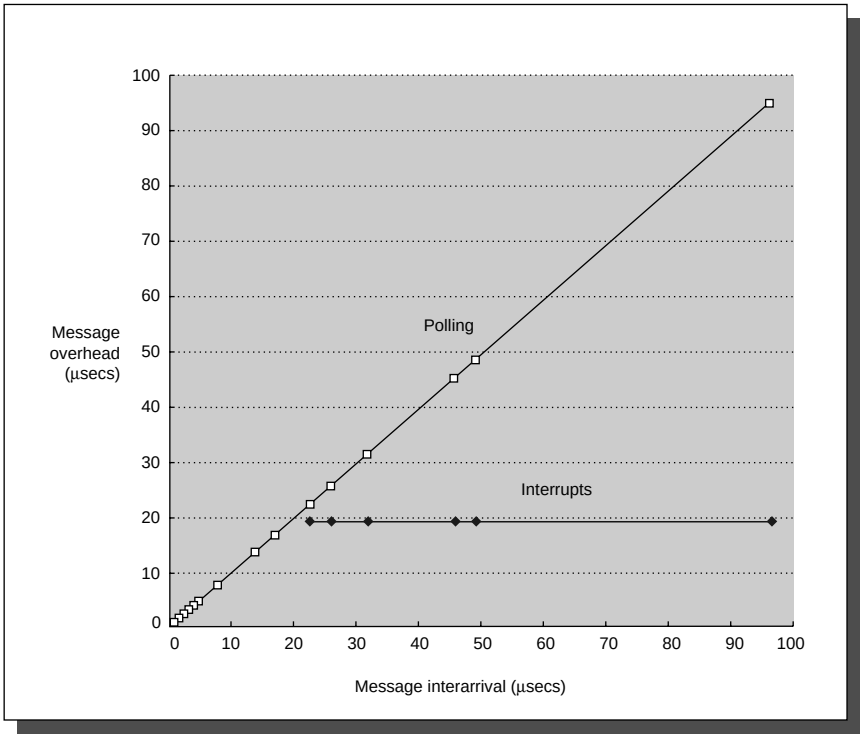

FIGURE 7.8 Message overhead versus message interarrival times for the CM-5. Liu and Culler [1994] took these measurements.

Given the parameters above, we want to avoid enabling and disabling interrupts, since the cost of invoking the kernel is large relative to the cost of receiving messages. The CM-5 uses the following scheme: Have interrupts enabled at all times, but on an interrupt the routine will poll for incoming messages before returning to the interrupted program. The virtue of this scheme is that it works well no matter what the load. When messages are arriving slowly, the overhead cost should be that of the interrupt code; when they arrive quickly, the cost should be that of polling, since the interrupt code will not return until all the messages have been received.

■

When selecting the network interface hardware, where to plug it into the machine, and how to interface to the software, try to follow these guidelines:

- Avoid invoking the operating system in the common case.
- Minimize the number of times operating at uncached memory speeds to interact with the network interface (such as to check status).

7.4 Interconnection Network Media

There is an old network saying: Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed—you can't bribe God.

David Clark, MIT

Just as there is a memory hierarchy, there is a hierarchy of media to interconnect computers that varies in cost, performance, and reliability. Network media have another figure of merit, the maximum distance between nodes. This section covers three popular examples, and Figure 7.9 illustrates them.

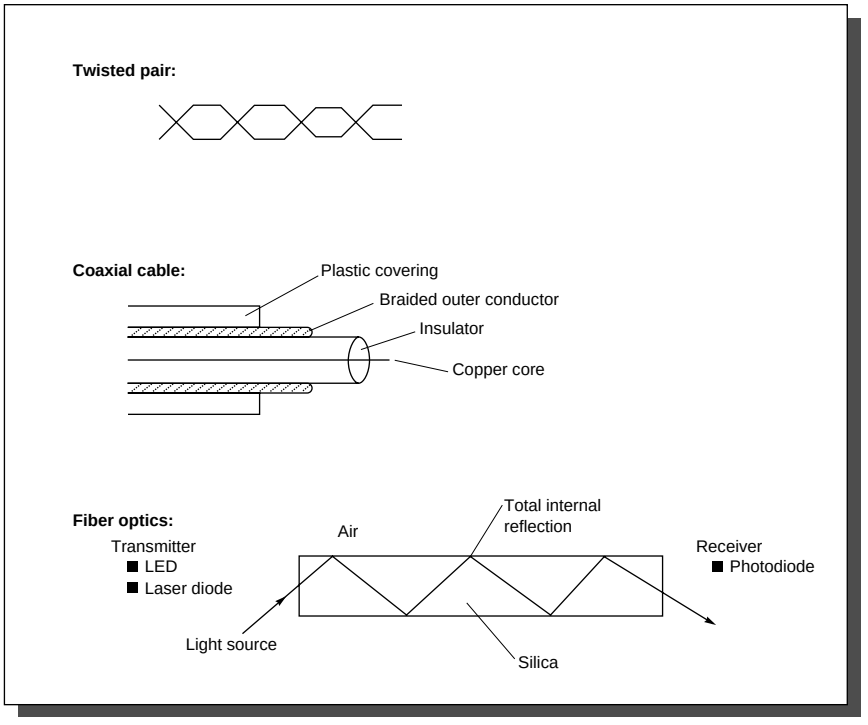

FIGURE 7.9 Three network media. (From a presentation by David Culler of U.C. Berkeley.)

The first medium is *twisted pairs* of copper wires. These are two insulated wires, each about 1 mm thick. They are twisted together to reduce electrical interference, since two parallel lines form an antenna but a twisted pair does not. As they can transfer a few megabits per second over several kilometers without amplification,

twisted pair were the mainstay of the telephone system. Telephone companies bundled together (and sheathed) many pairs coming into a building. Twisted pairs can also offer tens of megabits per second of bandwidth over shorter distances, making them plausible for LANs.

Coaxial cable was developed for the cable television companies to deliver a higher rate over a few kilometers. To offer high bandwidth and good noise immunity, a single stiff copper wire is surrounded by insulating material, and then the insulator is surrounded by cylindrical conductor, often woven as a braided mesh. A 50-ohm baseband coaxial cable delivers 10 megabits per second over a kilometer.

Connecting to this heavily insulated media is more challenging. The original technique was a *T junction*: the cable is cut in two and a connector is inserted that reconnects the cable and adds a third wire to a computer. A less invasive solution is a *vampire tap*: a hole of precise depth and width is first drilled into the cable, terminating in the copper core. A connector is then screwed in without having to cut the cable.

As the supply of copper has dwindled and to keep up with the demands of bandwidth and distance, it became clear that the telephone company would need to find new media. The solution could be more expensive provided that it offered much higher bandwidth and that supplies were plentiful. The answer was to replace copper with plastic and electrons with photons. *Fiber optics* transmits digital data as pulses of light: for example, light might mean 1 and no light might mean 0.

A fiber optic network has three components:

1. the transmission medium, a fiber optic cable;
2. the light source, an LED or laser diode;
3. the light detector, a photodiode.

Note that unlike twisted pairs or coax, fibers are one-way, or *simplex*, media. A two-way, or *full duplex*, connection between two nodes requires two fibers.

Since light is bent or refracted at interfaces, it can slowly be spread out as it travels down the cable unless the diameter of the cable is limited to one wavelength of light; then it transfers in a straight line. Thus fiber optic cables are of two forms:

1. *Multimode fiber*—Allows the light to be dispersed and uses inexpensive LEDs as a light source. It is useful for transmissions up to 2 kilometers and in 1995 transmits up to 600 megabits per second.
2. *Single-mode fiber*—This single-wavelength fiber requires more expensive laser diodes for light sources and currently transmits gigabits per second for hundreds of kilometers, making it the medium of choice for telephone companies.

Although single-mode fiber is a better transmitter, it is much more difficult to attach connectors to single-mode; it is less reliable and more expensive, and the cable itself has restrictions on the degree it can be bent. Hence when ease of connection is more important than very long distance, such as in a LAN, multimode fiber is likely to be popular.

Connecting fiber optics to a computer is more challenging than connecting cable. The vampire tap solution of cable fails because it loses light. There are two forms of T-boxes:

1. Taps are fused onto the optical fiber. Each tap is passive, so a failure cuts off just a single computer.
2. In an active repeater, light is converted to electrical signals, sent to the computer, converted back to light, and then sent down the cable. If an active repeater fails, it blocks the network.

In both cases, fiber optics has the additional cost of optical-to-electrical and electrical-to-optical conversion as part of the computer interface.

The product of the bandwidth and maximum distance forms a single figure of merit: gigabit-kilometers per second. According to Desurvire [1992], since 1975 optical fibers have increased transmission capacity by tenfold every four years by this measure.

Figure 7.10 shows the typical distance, bandwidth, and cost of the three media. Compared to the electrical media, fiber optics are more difficult to tap, have more expensive interfaces, go for longer distances, and are less likely to experience degradation due to noise.

Media	Bandwidth	Maximum distance	Bandwidth × distance	Cost per meter	Cost for termination	Labor cost to install	Cost per computer interface
Twisted pair copper wire	1 Mb/sec (20 Mb/sec)	2 km (0.1 km)	0.02 Gb-km/sec	$0.23	$4.60	$2.00	≈$2
Coaxial cable	10 Mb/sec	1 km	0.01 Gb-km/sec	$1.64	$220.00	$15.00	≈$5
Multimode optical fiber	600 Mb/sec	2 km	1.20 Gb-km/sec	$1.03	$11.80	$10.00	≈$1000
Single-mode optical fiber	2000 Mb/sec	100 km	200.00 Gb-km/sec	$1.64	$23.90	$10.00	≈$1000

FIGURE 7.10 Figures of merit for several network media in 1995. The coaxial cable is the Thick Net Ethernet standard (see Figure 7.9) using a vampire tap for termination. Twisted-pair Ethernet lowers cost by using the media in the first row. Since an optical fiber is a one-way, or simplex, media, the costs per meter and for termination in this figure are for two strands to supply two-way, or full duplex, communication. The major costs for fiber are the electrical-optical interfaces.

Let's compare these media in an example.

EXAMPLE Suppose you have 100 magnetic tapes, each containing 10 GB. Assume that you have enough tape readers to keep any network busy. How long will it take to transmit the data over a distance of one kilometer using each of the media in Figure 7.10? How do they compare to delivering the tapes by car?

ANSWER The amount of data is 1000 GB. The time for each medium is given below:

$$\text{Twisted pair} = \frac{1000 \times 1024 \times 8 \text{ Mb}}{1 \text{ Mb/sec}} = 8{,}192{,}000 \text{ secs} = 95 \text{ days}$$

$$\text{Coaxial cable} = \frac{1000 \times 1024 \times 8 \text{ Mb}}{10 \text{ Mb/sec}} = 819{,}200 \text{ secs} = 9.5 \text{ days}$$

$$\text{Multimode fiber} = \frac{1000 \times 1024 \times 8 \text{ Mb}}{600 \text{ Mb/sec}} = 13{,}653 \text{ secs} = 3.8 \text{ hours}$$

$$\text{Single-mode fiber} = \frac{1000 \times 1024 \times 8 \text{ Mb}}{2000 \text{ Mb/sec}} = 4096 \text{ secs} = 1.1 \text{ hours}$$

$$\text{Car} = \text{Time to load car} + \text{Transport time} + \text{Time to unload car}$$

$$= 300 \text{ secs} + \frac{1 \text{ km}}{50 \text{ kph}} + 300 \text{ secs} = 300 \text{ secs} + 72 \text{ secs} + 300 \text{ secs}$$

$$= 672 \text{ secs} = 11.2 \text{ min}$$

A car filled with tapes is a high-bandwidth medium! ■

7.5 Connecting More Than Two Computers

Thus far we have discussed two computers communicating over private lines, but what makes interconnection networks interesting is the ability to connect hundreds of computers together. And what makes them more interesting also makes them more challenging to build.

Shared versus Switched Media

Certainly the simplest way to connect multiple computers is to have them share a single interconnection medium, just as I/O devices share a single I/O bus. The most popular LAN, Ethernet, is simply a bus that can be shared by hundreds of computers.

Given that the medium is shared, there must be a mechanism to coordinate the use of the shared medium so that only one message is sent at a time. If the network is small, it may be possible to have an additional central arbiter to give

permission to send a message. (Of course, this leaves open the question of how the nodes talk to the arbiter.)

Centralized arbitration is impractical for networks with a large number of nodes spread out over a kilometer, so we must distribute arbitration. A node first listens to make sure it doesn't send a message while another message is on the network. If the interconnection is idle, the node tries to send. Of course, some other node may decide to send at the same instant. When two nodes send at the same time, it is called a *collision*. Let's assume that the network interface can detect any resulting collisions by listening to what is sent to hear if the data were garbled by other data appearing on the line. Listening to avoid and detect collisions is called *carrier sensing and collision detection.*

To avoid repeated head-on collisions, each node whose message was garbled waits (or "backs off") a random time before resending. Subsequent collisions result in exponentially increasing time between attempts to retransmit. Although this approach is not guaranteed to be fair—some subsequent node may transmit while those that collided are waiting—it does control congestion. If the network does not have high demand from many nodes, this simple approach works well. Under high utilization, performance degrades since the medium is shared.

Shared media have some of the same advantages and disadvantages as buses: they are inexpensive, but they have limited bandwidth. The alternative to sharing the media is to have a dedicated line to a switch that in turn provides a dedicated line to all destinations. Figure 7.11 shows the potential bandwidth improvement of switches: *Aggregate bandwidth* is many times that of the single shared medium.

Switches allow communication directly from source to destination, without intermediate nodes to interfere with these signals. Such *point-to-point* communication is faster than a line shared between many nodes because there is no arbitration and the interface is simpler electrically. Of course, it does pay the added latency of going through the switch.

Every node of a shared line will see every message, even if it is just to check to see whether or not the message is for that node, so this style of communication is sometimes called *broadcast* to contrast it with point-to-point. The shared medium makes it easy to broadcast a message to every node, and even to broadcast to subsets of nodes, called *multicasting*.

Switches allow multiple pairs of nodes to communicate simultaneously, giving these interconnections much higher *aggregate* bandwidth than the speed of a shared link to a node. Switches also allow the interconnection network to scale to a very large number of nodes. Switches are called *data switching exchanges*, *multistage interconnection networks*, or even *interface message processors* (*IMPs*). Depending on the distance of the node to the switch, the network medium is either copper wire or optical fiber.

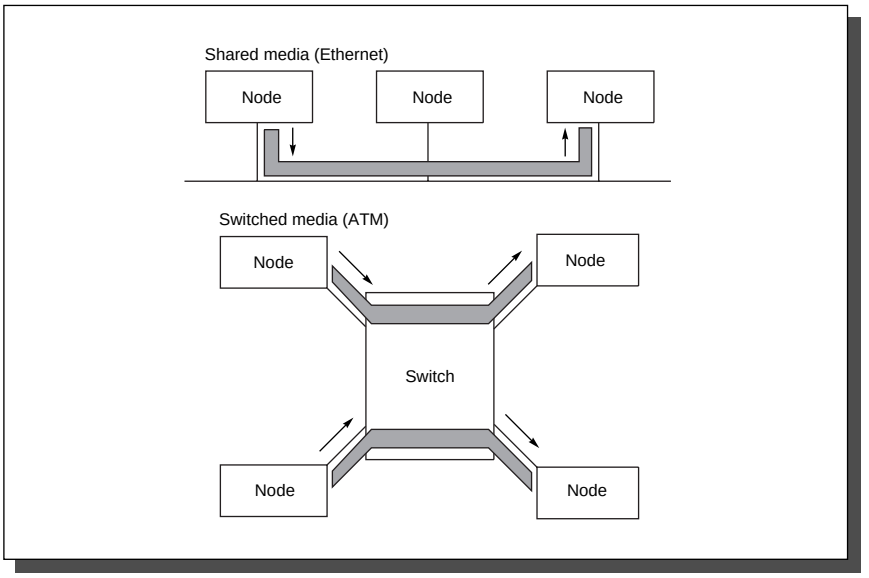

FIGURE 7.11 Shared medium versus switch. Ethernet is a shared medium and ATM is a switch-based medium. All nodes on the Ethernet must share the 10 Mb/sec interconnection, but switches like ATM can support multiple 155 Mb/sec transfers simultaneously.

EXAMPLE Compare 16 nodes connected three ways: a single 10 Mb/sec coaxial cable; a switch connected via twisted pairs, each running at 10 Mb/sec; and a switch connected via optical fibers, each running at 100 Mb/sec. The single coax is 500 meters long, and the average length of each segment to a switch is 50 meters. Both switches can support the full bandwidth, with the slower version costing $10,000 and the faster version costing $15,000. Assume each switch adds 50 microseconds to the latency. Calculate the aggregate bandwidth, transport latency, and cost of each alternative. Assume the average message size is 125 bytes.

ANSWER The aggregate bandwidth of each example is the simplest calculation: 10 Mb/sec for the single coax; 16×10, or 160 Mb/sec for the switched twisted pairs; and 16×100, or 1600 Mb/sec for the switched optical fibers.

The transport time is

$$\text{Transport time} = \text{Time of flight} + \frac{\text{Message size}}{\text{Bandwidth}}$$

For coax we just plug in the distance, bandwidth, and message size:

$$\begin{aligned}\text{Transport time}_{\text{coax}} &= \frac{500/1000 \times 10^6}{0.5 \times 299{,}792.5}\ \mu\text{secs} + \frac{125 \times 8}{10}\ \mu\text{secs} \\ &= 3.3\ \mu\text{secs} + 100\ \mu\text{secs} \\ &= 103.3\ \mu\text{secs}\end{aligned}$$

For the switches, the distance is twice the average segment, since there is one segment from the sender to the switch and one from the switch to the receiver. We must also add the latency for the switch.

$$\begin{aligned}\text{Transport time}_{\text{tp}} &= 2 \times \left(\frac{50/1000 \times 10^6}{0.5 \times 299{,}792.5}\right)\ \mu\text{secs} + 50\ \mu\text{secs} + \frac{125 \times 8}{10}\ \mu\text{secs} \\ &= 0.7\ \mu\text{secs} + 50\ \mu\text{secs} + 100\ \mu\text{secs} \\ &= 150.7\ \mu\text{secs} \\ \text{Transport time}_{\text{fiber}} &= 2 \times \left(\frac{50/1000 \times 10^6}{0.5 \times 299{,}792.5}\right)\ \mu\text{secs} + 50\ \mu\text{secs} + \frac{125 \times 8}{100}\ \mu\text{secs} \\ &= 0.7\ \mu\text{secs} + 50\ \mu\text{secs} + 10\ \mu\text{secs} \\ &= 60.7\ \mu\text{secs}\end{aligned}$$

Figure 7.12 shows the costs of each option, based on Figure 7.10. We assumed that the switches included the termination and interfaces. Since the media is connected to both the nodes and to the switch, we doubled the labor costs.

	Coax	Twisted pair	Fiber optic
Termination	$3520	$74	$189
Labor	$240	$64	$320
Node interfaces	$80	$32	$16,000
Media	$820	$184	$824
Switch		$10,000	$15,000
Total	$4660	$10,354	$32,333

FIGURE 7.12 Costs of single coax, twisted pair using switch, and fiber optics using switch, using costs in Figure 7.10 (page 578).

The high costs of the thick coaxial cable and vampire taps, illustrated in this example, have led to the use of twisted pairs for shorter distance LANs. Although the continuing silicon revolution will lower the price of the switch, the challenge for the optical fiber is to bring down the cost of the electrical-optical interfaces. ■

Switches allow communication to harvest the same rapid advance from silicon as have processors and main memory. Whereas the switches from telecommunications companies were once the size of mainframe computers, today we see single-chip switches in MPPs. Just as single-chip processors led to processors replacing logic in a surprising number of places, single-chip switches will increasingly replace buses and shared media interconnection networks.

Switch Topology

The number of different topologies that have been discussed in publications would be difficult to count, but the number that have been used commercially is just a handful, with MPP designers being the most visible and imaginative. MPPs have used regular topologies to simplify packaging and scalability. The topologies of LANs and WANs are more haphazard, having more to do with the challenges of long distance or simply the connection of equipment purchased over several years.

Figure 7.13 illustrates two of the popular switch organizations, with the path from node P_0 to node P_6 shown in gray in each topology. A fully connected, or *crossbar,* interconnection allows any node to communicate with any other node in one pass through the interconnection. An *Omega* interconnection uses less hardware than the crossbar interconnection ($n/2 \log_2 n$ vs. n^2 switches), but contention is more likely to occur between messages, depending on the pattern of communication. The term *blocking* is used to describe this form of contention. For example, in the Omega interconnection in Figure 7.13 a message from P_1 to P_7 is blocked while waiting for a message from P_0 to P_6. Of course, if two nodes try to send to the same destination—both P_0 and P_1 send to P_6—there will be contention for that link, even in the crossbar.

Another switch is based on a tree with bandwidth added higher in the tree to match the requirements of common communications patterns. This topology, commonly called a *fat tree*, is shown in Figure 7.14. Interconnections are normally drawn as graphs, with each arc of the graph representing a link of the communication interconnection, with nodes shown as black squares and switches shown as shaded circles. This figure shows that there are multiple paths between any two nodes; for example, between node 0 and node 8 there are four paths. If messages are randomly assigned to different paths, communication can take advantage of the full bandwidth of the fat-tree topology.

Thus far the switch has been separate from the processor and memory and assumed to be located in a central location. Looking inside this switch we see many smaller switches. The term *multistage switch* is sometimes used to refer to centralized units to reflect the multiple steps that a message may travel before it reaches a computer. Instead of centralizing these small switching elements, an alternative is to place one small switch at every computer, yielding a distributed switching unit.

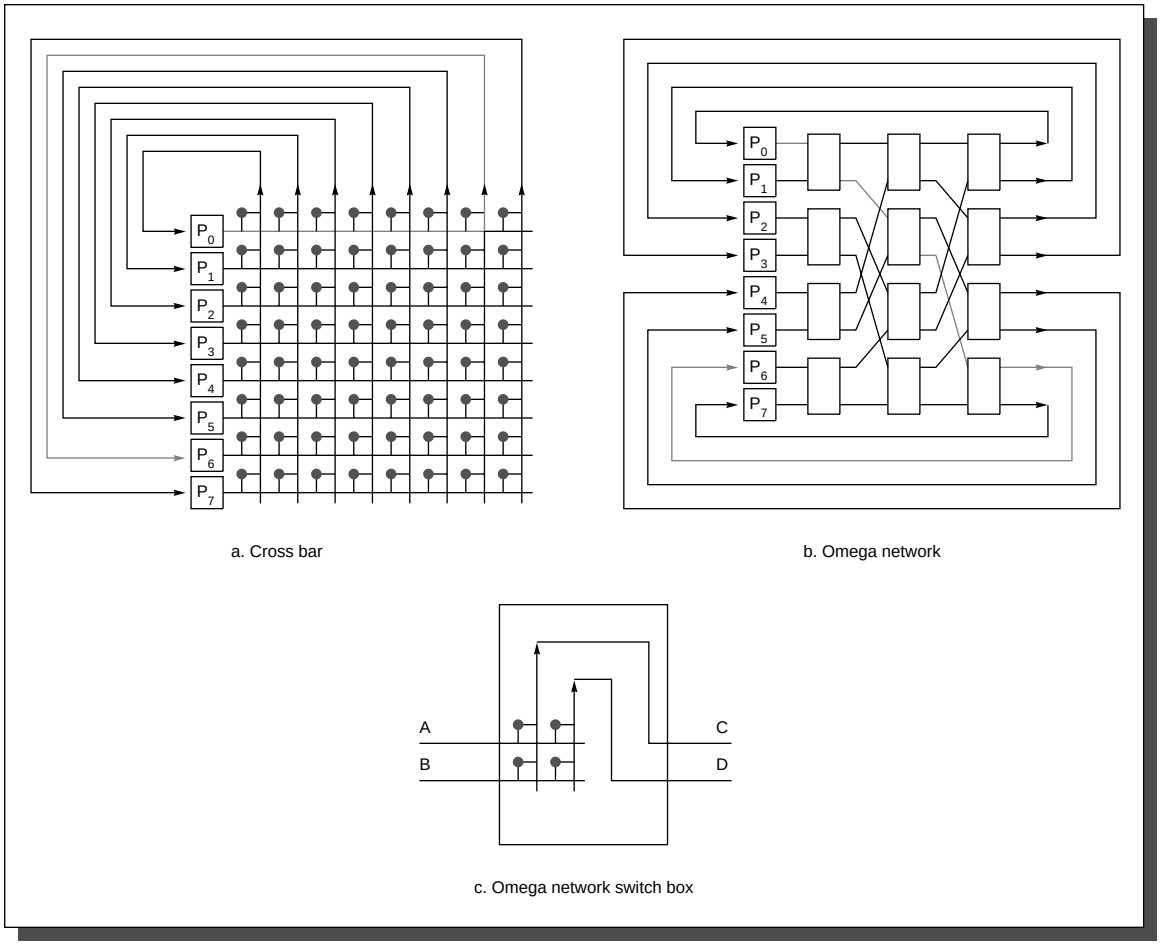

FIGURE 7.13 Popular switch topologies for eight nodes. The links are unidirectional; data come in at the left and exit out the right link. The switch box in (c) can pass A to C and B to D or B to C and A to D. The crossbar uses n^2 switches, where n is the number of processors, while the Omega network uses $n/2 \log_2 n$ of the large switch boxes, each of which is logically composed of four of the smaller switches. In this case the crossbar uses 64 switches versus 12 switch boxes or 48 switches in the Omega network. The crossbar, however, can simultaneously route any permutation of traffic pattern between processors. The Omega network cannot.

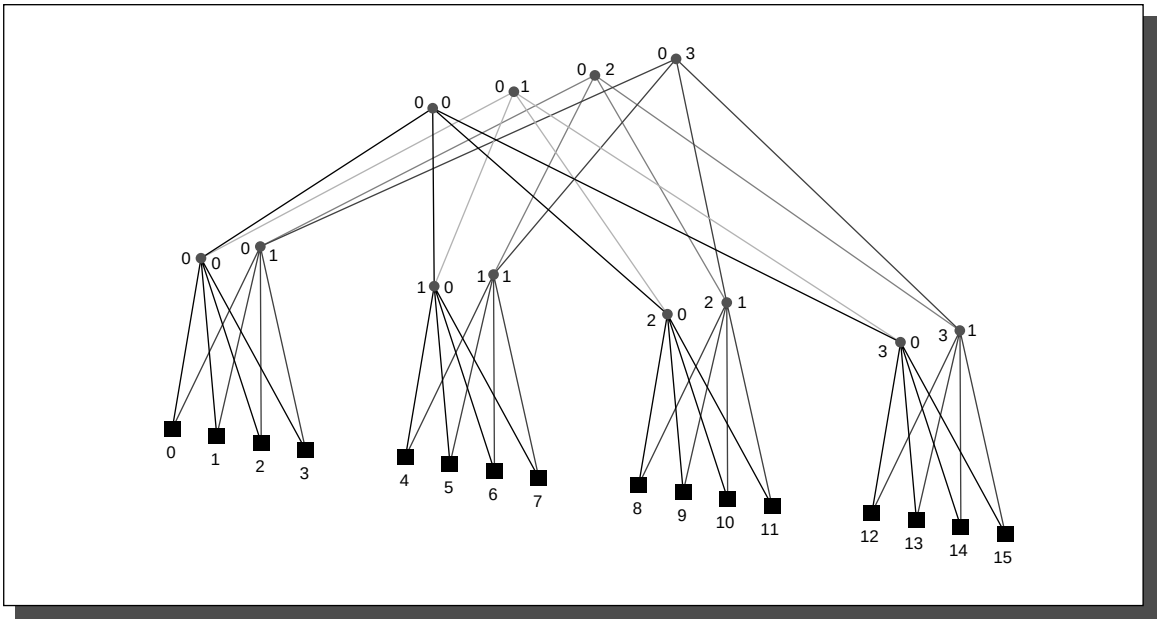

FIGURE 7.14 A fat-tree topology for 16 nodes. The shaded circles are switches, and the squares at the bottom are processor-memory nodes. A simple 4-ary tree would only have the links at the front of the figure; that is, the tree with the root labeled 0,0. This three-dimensional view suggests the increase in bandwidth via extra links at each level over a simple tree, so bandwidth between each level of a fat tree is normally constant rather than being reduced by a factor of four as in a 4-ary tree. Multiple paths and random routing give it the ability to route common patterns well, which ensures no single pattern from a broad class of communication patterns will do badly. In the CM-5 fat-tree implementation, the switches have four downward connections and two or four upward connections; in this figure the switches have two upward connections.

Given a distributed switch, the question is how to connect the switches together. Figure 7.15 shows that a low-cost alternative to full interconnection is a network that connects a sequence of nodes together. This topology is called a *ring*. Since some nodes are not directly connected, some messages will have to hop along intermediate nodes until they arrive at the final destination. Unlike shared lines, a ring is capable of many simultaneous transfers: the first node can send to the second at the same time as the third node can send to the fourth, for example. Rings are not quite as good as this sounds because the average message must travel through $n/2$ switches, where n is the number of nodes. To first order, a ring is like a pipelined bus: on the plus side are point-to-point links, and on the minus side are “bus repeater” delays.

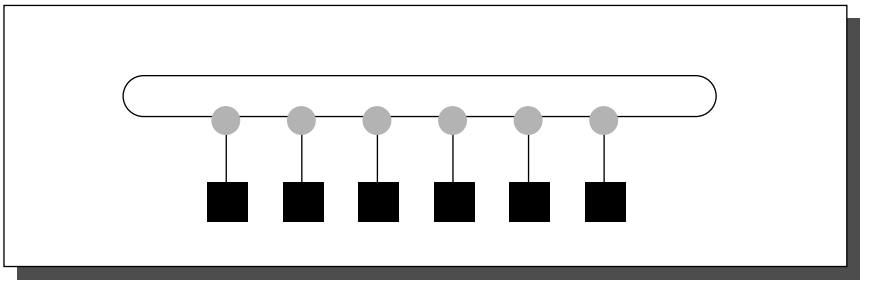

FIGURE 7.15 A ring network topology.

One variation of rings used in local area networks is the *token ring*. To simplify arbitration, a single slot, or *token*, is passed around the ring to determine which node is allowed to send a message; a node can send only when it gets the token. (A token is simply a special bit pattern.) In this section we will evaluate the ring as a topology with more bandwidth rather than one that may be simpler to arbitrate than a long shared medium.

A straightforward but expensive alternative to a ring is to have a dedicated communication link between every switch. The tremendous improvement in performance of fully connected switches is offset by the enormous increase in cost, typically going up with the square of the number of nodes. This cost inspires designers to invent new topologies that are between the cost of rings and the performance of fully connected networks. The evaluation of success depends in large part on the nature of the communication in the interconnection network. Real machines frequently add extra links to these simple topologies to improve performance and reliability. Figure 7.16 illustrates three popular topologies for MPPs.

One popular measure for MPP interconnections, in addition to the ones covered in section 7.2, is the *bisection bandwidth*. This measure is calculated by dividing the interconnect into two roughly equal parts, each with half the nodes. You then sum the bandwidth of the lines that cross that imaginary dividing line. For fully connected interconnections the bisection bandwidth is $(n/2)^2$, where n is the number of nodes.

Since some interconnections are not symmetric, the question arises as to where to draw the imaginary line when bisecting the interconnect. Bisection bandwidth is a worst-case metric, so the answer is to choose the division that makes interconnection performance worst; stated alternatively, calculate all possible bisection bandwidths and pick the smallest. Figure 7.17 summarizes these different topologies using bisection bandwidth and the number of links for 64 nodes.

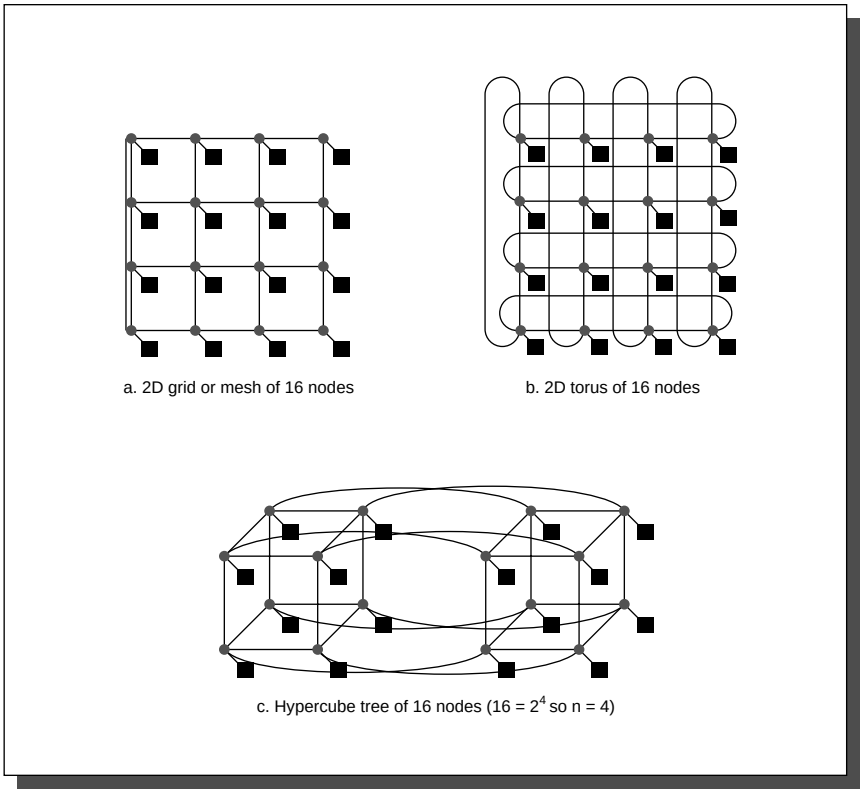

FIGURE 7.16 Network topologies that have appeared in commercial MPPs. The shaded circles represent switches, and the black squares represent nodes. Even though a switch has many links, generally only one goes to the node. Frequently these basic topologies have been supplemented with extra arcs to improve performance and reliability. For example, the switches in the left and right columns of the 2D grid are connected together using the unused ports on each switch to form the 2D torus. The Boolean hypercube topology is an *n*-dimensional interconnect for 2^n nodes, requiring *n* ports per switch (plus one for the processor), and thus *n* nearest neighbor nodes.

Evaluation category	Bus	Ring	2D torus	6-cube	Fully connected
Performance					
Bisection bandwidth	1	2	16	32	1024
Cost					
Ports per switch	NA	3	5	7	64
Total number of lines	1	128	192	256	2080

FIGURE 7.17 Relative cost and performance of several interconnects for 64 nodes. The bus is the standard reference at unit cost, and of course there can be more than one data line along each link between nodes. Note that any network topology that scales the bisection bandwidth linearly must scale the number of interconnection lines faster than linearly. Figure 7.13a on page 584 is an example of a fully connected network.

EXAMPLE Let's examine the difference between the bisection bandwidths of each topology in Figure 7.17 for 64 nodes. Assume all-to-all communication: each node does a single transfer to every other node. We simplify the communication cost model for this example: it takes one time unit to go from switch to switch, and there is no cost in or out of the processor. Assuming every link of every interconnect is the same speed and that a node can send as many messages as it wants at a time, how long does it take for complete communication? (See Exercise 7.8 for a more realistic version of this example.)

ANSWER For each node to send a message to every other node, we need to send 64×63 or 4032 messages. Here are the cases in increasing order of difficulty of explanation:

- *Bus*—Transfers are done sequentially, so it takes 4032 time units.
- *Fully connected*—All transfers are done in parallel, taking one time unit.
- *Ring*—This is easiest to see step by step. In the first step each node sends a message to the node with the next higher address, with node 63 sending to node 0. This takes one step for all 64 transfers to the nearest neighbor. The second step sends to the node address + 2 modulo 64. Since this goes through two links, it takes two time units. It would seem that this would continue until we send to the node address + 63 modulo 64 taking 63 time units, but remember that these are bidirectional links. Hence, sending from node 1 to node 1 + 63 modulo 64 = 0 takes just one time unit because there is a link connecting them together. Then the calculation for the ring is

$$\begin{aligned} \text{Time}_{\text{Ring}} &= 1 + 2 + \ldots + 31 + 32 + 31 + \ldots + 2 + 1 \\ &= \frac{31 \times 32}{2} + 32 + \frac{31 \times 32}{2} = 496 + 32 + 496 \\ &= 1024 \end{aligned}$$

- *2D torus*—There are eight rows and eight columns in our torus of 64 nodes. Remember that the top and bottom rows of a torus are just one link away, as are the leftmost and rightmost columns. This allows us to treat the communication as we did the ring, in that there are no special cases at the edges. Let's first calculate the time to send a message to all the nodes in the same row. This time is the same as a ring with just eight nodes:

$$\begin{aligned}\text{Time}_{\text{Row}} &= 1 + 2 + 3 + 4 + 3 + 2 + 1 \\ &= \frac{3 \times 4}{2} + 4 + \frac{3 \times 4}{2} = 6 + 4 + 6 \\ &= 16\end{aligned}$$

To send a message to all the elements in the row below, all eight messages must first take one time unit to get to that row. The time for the eight messages to get to the proper node within that row is the same as the time to send a message to all elements of a row:

$$\text{Time}_{\text{Row below}} = 8 \times 1 + \text{Time}_{\text{Row}}$$

This can be generalized as the time it takes to send eight messages to each row plus the time to distribute the messages within a row:

$$\begin{aligned}\text{Time}_{\text{2D}} &= \text{Time}_{\text{Row}} + (8 \times 1 + \text{Time}_{\text{Row}}) + (8 \times 2 + \text{Time}_{\text{Row}}) + \ldots \\ &\quad + (8 \times 4 + \text{Time}_{\text{Row}}) + \ldots + (8 \times 1 + \text{Time}_{\text{Row}}) \\ &= \text{Time}_{\text{Row}} + (8 \times 1 + 8 \times 2 + 8 \times 3 + 8 \times 4 + 8 \times 3 + 8 \times 2 + 8 \times 1) + 7 \times \text{Time}_{\text{Row}} \\ &= 8 \times \text{Time}_{\text{Row}} + 8 \times (1 + 2 + 3 + 4 + 3 + 2 + 1) \\ &= 256\end{aligned}$$

This is only sending one message at a time per node, even though each node has multiple links. The communication pattern suggested above first uses vertical and then horizontal links, using only half the potential interconnection bandwidth. By carefully selecting pairs of communications that have the same number of vertical hops as the other has horizontal hops and vice versa, the time for complete communication can be cut approximately in half.

- *6-cube*—The number of nodes at each distance in a 6-cube can be found from Pascal's Triangle: 6 at distance 1, 15 at 2, 20 at 3, 15 at 4, 6 at 5, and 1 at distance 6. The number of hops for a node to send to all other nodes is

  $(1 \times 6) + (2 \times 15) + (3 \times 20) + (4 \times 15) + (5 \times 6) + (6 * 1) = 6 + 30 + 60 + 60 + 30 + 6 = 192$

  Since every node must send this message, the total number required is $64 \times 192 = 12{,}228$. Since the interconnection network has $64 \times (6/2) = 192$ links, it will take $12{,}228/192 = 64$ cycles to complete all hops.

Figure 7.18 summarizes the calculations from this example.

Evaluation category	Bus	Ring	2D torus	6-cube	Fully connected
Time all-to-all	4032	1024	256	64	1
Time north & east	112	8	1	1	1

FIGURE 7.18 Summary of the communication times for all-to-all and for nearest northern and eastern neighbors calculated in the surrounding examples.

■

EXAMPLE A common communication pattern in scientific programs is to consider the nodes as elements of a two-dimensional array and then have communication to the nearest neighbor in a given direction. (This is sometimes called NEWS communication, standing for north, east, west, and south, the directions on the compass.) Map an eight-by-eight array onto the 64 nodes in each topology, and assume every link of every interconnect is the same speed. How long does it take for each node to send one message to its northern neighbor and one to its eastern neighbor? Ignore nodes that have no northern or eastern neighbors.

ANSWER In this case we want to send $2 \times (64 - 8)$, or 112, messages. Here are the cases, again in increasing order of difficulty of explanation:

- *Bus*—The placement of the eight-by-eight array makes no difference for the bus, since all nodes are equally distant. The 112 transfers are done sequentially, taking 112 time units.
- *Fully connected*—Again the nodes are equally distant; all transfers are done in parallel, taking one time unit.
- *Ring*—Here the nodes are differing distances. Assume the first row of the array is placed on nodes 0 to 7, the second row on nodes 8 to 15, and so on. It takes just one time unit to send to the eastern neighbor, for this is a send from node n to node $n + 1$. The northern neighbor is exactly eight nodes away in this scheme, so it takes eight time units for each node to send to its northern neighbor. The ring total is nine time units.
- *2D torus*—There are eight rows and eight columns in our grid of 64 nodes, which is a perfect match to the NEWS communication. It takes just two time units to send to the northern and eastern neighbors.
- *6-cube*—It is possible to place the array so that it will take just two time units for this communication pattern, as in the case of the 2D grid.

Figure 7.18 also summarizes the calculations from this example. ■

The simple analysis of interconnection networks in this section ignores several important practical considerations in the construction of an interconnection network. First, these three-dimensional drawings must be mapped onto chips, boards, and cabinets that are essentially two-dimensional media, often tree-like. For example, due to the fixed height of cabinets, an *n*-node Intel Paragon uses an $n/16 \times 16$ rectangular grid rather than the ideal of $\sqrt{n} \times \sqrt{n}$. Another consideration is the internal speed of the switch: if it is fixed, then more links per switch means lower bandwidth per link, potentially affecting the desirability of different topologies. Yet another consideration is that the latency through a switch depends on the complexity of the routing pattern, which in turn depends on the topology.

Topologies that appear elegant when sketched on the blackboard may look awkward when constructed from chips, cables, boards, and boxes. The bottom line is that quality of implementation matters more than topology. To put these topologies in perspective, Figure 7.19 lists those used in commercial MPPs.

Institution	Name	Number of nodes	Basic topology	Data bits/link	Network clock rate	Peak BW/link (MB/sec)	Bisection (MB/sec)	Year
Thinking Machines	CM-2	1024 to 4096	12-cube	1	7 MHz	1	1024	1987
nCube	nCube/ten	1 to 1024	10-cube	1	10 MHz	1.2	640	1987
Intel	iPSC/2	16 to 128	7-cube	1	16 MHz	2	345	1988
Maspar	MP-1216	32 to 512	2D grid + multistage Omega	1	25 MHz	3	1300	1989
Intel	Delta	540	2D grid	16	40 MHz	40	640	1991
Thinking Machines	CM-5	32 to 2048	Multistage fat tree	4	40 MHz	20	10,240	1991
Meiko	CS-2	32 to 1024	Multistage fat tree	8	70 MHz	50	50,000	1992
Intel	Paragon	4 to 2048	2D grid	16	100 MHz	175	6400	1992
IBM	SP-2	2 to 512	Multistage fat tree	8	40 MHz	40	20,480	1993
Cray Research	T3D	16 to 2048	3D torus	16	150 MHz	300	76,800	1993

FIGURE 7.19 Characteristics of interconnections of some commercial MPPs. The bisection bandwidth is given for the largest machine. The 2D grid of the Intel Delta is 16 rows by 35 columns. The fat-tree topology of the CM-5 is restricted in the lower two levels, hence the lower bandwidth in the bisection. Note that the Cray T3D has two processors per node and the Intel Paragon has from two to as many as four processors per node.

Connection-Oriented versus Connectionless Communication

Before computers arrived on the scene, the telecommunications industry allowed communication around the world. An operator sets up a *connection* between a caller and a callee, and once the connection is established, a conversation can continue for hours. To share transmission lines over long distances, the telecommunications industry uses switches to multiplex several conversations on the same lines. Since audio transmissions have relatively low bandwidth, the solution was to divide the bandwidth of the transmission line into a fixed number of slots, with each slot assigned to a conversation. This technique is called *frequency-division multiplexing*.

Although a good match for voice, frequency-division multiplexing is inefficient for sending data. The problem is that the time slot is dedicated to the conversation whether or not there is anything being said. Hence the long distance lines are "busy" based on the *number* of conversations, and not on the *amount* of information being sent at a particular time. An alternative style of communication is called *connectionless*, where each package is routed to the destination by looking at its address. The postal system is a good example of connectionless communication.

Closely related to the idea of connection versus connectionless communication are the terms *circuit switching* and *packet switching*. Circuit switching is the traditional way to offer a connection-based service. A circuit is established from source to destination to carry the conversation, reserving bandwidth until the circuit is broken. The alternative to circuit-switched transmission is to divide the information into *packets*, or *frames*, with each packet including the destination of the packet plus a portion of the information. Queuing theory in section 6.4 tells us that packets cannot use all of the bandwidth, but in general this *packet-switched* approach allows more use of the bandwidth of the medium and is the traditional way to support connectionless communication.

EXAMPLE Let's compare a single 100 Mbits/sec packet switched network with ten 10 Mbits/sec packet-switched networks. Assume that the mean size of a packet is 250 bytes, the arrival rate is 25,000 packets per second, and the interarrival times are exponentially distributed. What is the mean response time for each alternative? What is the intuitive reason behind the difference?

ANSWER From section 6.4 in the prior chapter, we can use an M/M/1 queue to calculate the mean response time for the single fast network:

$$\text{Service rate} = \frac{100 \times 10^6}{250 \times 8} = \frac{100 \times 10^6}{2000} = 50{,}000 \text{ packets per second}$$

$$\text{Time}_{\text{server}} = \frac{1}{50{,}000} = 0.00002 \text{ secs} = 20 \text{ µsecs}$$

$$\text{Utilization} = \frac{\text{Arrival rate}}{\text{Service rate}} = \frac{25{,}000}{50{,}000} = 0.5$$

$$\text{Time}_{\text{queue}} = \text{Time}_{\text{server}} \times \frac{\text{Server utilization}}{(1 - \text{Server utilization})} = 20 \text{ µsecs} \times \frac{0.5}{1 - 0.5} = 20 \times \frac{0.5}{0.5} = 20 \text{ µsecs}$$

$$\text{Mean response time} = \text{Time}_{\text{queue}} + \text{Time}_{\text{server}} = 20 + 20 = 40 \text{ µsecs}$$

The 10 slow networks can be modeled by an M/M/m queue, and the appropriate formulas are found in section 6.7:

$$\text{Service rate} = \frac{10 \times 10^6}{250 \times 8} = \frac{10 \times 10^6}{2000} = 5000 \text{ packets per second}$$

$$\text{Time}_{\text{server}} = \frac{1}{5000} = 0.0002 \text{ secs} = 200 \text{ µsecs}$$

$$\text{Utilization} = \frac{\text{Arrival rate}}{m \times \text{Service rate}} = \frac{25{,}000}{10 \times 5000} = \frac{25{,}000}{50{,}000} = 0.5$$

$$\text{Time}_{\text{queue}} = \text{Time}_{\text{server}} \times \frac{\text{Server utilization}}{m \times (1 - \text{Server utilization})} = 200 \text{ µsecs} \times \frac{0.5}{10 \times (1 - 0.5)} = 20 \times \frac{0.5}{0.5} = 20 \text{ µsecs}$$

$$\text{Mean response time} = \text{Time}_{\text{queue}} + \text{Time}_{\text{server}} = 20 + 200 = 220 \text{ µsecs}$$

The intuition is clear from the results: the service time is much less for the faster networks even though the queuing times are the same. This intuition is the argument for "statistical multiplexing" using packets; queuing times are not worse for a single faster network, and the latency for a single packet is much less. Stated alternatively, you get better latency when you use an unloaded fast network, and data traffic is bursty so it works. ■

Although connections are traditionally aligned with circuit switching, it is certainly possible to provide the user the appearance of a logical connection on top of a packet-switched network. TCP/IP, as we shall see in section 7.9, is a connection-oriented service that operates over packet-switched networks.

Routing: Delivering Messages

Given that the path between nodes may be difficult to navigate depending upon the topology, the system must be able to route the message to the desired node. Shared media has a simple solution: The message is broadcast to *all* nodes that share the media, and each node looks at an address within the message to see

whether the message is for that node. This routing also made it easy to broadcast one message to all nodes by reserving one address for everyone; broadcast is much harder to support in switch-based networks.

Switched media use three solutions for routing. In *source-based routing*, the message specifies the path to the destination. Since the network merely follows directions, it can be simpler. One alternative is the *virtual circuit*, whereby a circuit is established between source and destination, and the message simply names the circuit to follow. Another approach is a *destination-based routing*, where the message merely contains a destination address, and the switch must pick a path to deliver the message. Destination-based routing may be *deterministic* and always follow the same path, or it may be *adaptive*, allowing the network to pick different routes to avoid failures or congestion. Closely related to adaptive routing is *randomized routing*, whereby the network will randomly pick between several equally good paths to spread the traffic throughout the network, thereby avoiding hot spots.

Switches in wide area networks route messages using a *store-and-forward* policy; each switch waits for the full message to arrive in the switch before it is sent on to the next switch. The alternative is for the switch to examine the header, decide where to send the message, and then start forwarding it immediately without waiting for the rest of the message. This alternative is called either *cut-through* routing or *wormhole* routing and is popular in MPP networks. In wormhole routing, when the head of the message is blocked, the message stays strung out over the network, potentially blocking other messages. Cut-through routing lets the tail continue when the head is blocked, compressing the strung-out message into a single switch. Clearly, cut-through routing requires a buffer large enough to hold the largest packet, while wormhole routing needs only to buffer the piece of the packet that is sent between switches.

The advantage of both cut-through and wormhole routing over store-and-forward is that latency reduces from a function of the number of intermediate switches *multiplied* by the size of the packet to the time for the first part of the packet to negotiate the switches *plus* the transmission time.

EXAMPLE The CM-5 uses wormhole routing, with each switch buffer being just 4 bits per port. Compare efficiency of store-and-forward versus wormhole routing for a 128-node machine using a CM-5 interconnection sending a 16-byte payload. Assume each switch takes 0.25 microseconds and that the transfer rate is 20 MB/sec.

ANSWER Each switch in the CM-5 is one node of the 4-ary fat tree. The CM-5 interconnection for 128 nodes is four levels high, so a message goes through seven intermediate switches. Each CM-5 packet has four bytes of header information, so the length of this packet is 20 bytes. The time to transfer 20 bytes over one CM-5 link is

$$\frac{20}{20 \text{ MB/sec}} = 1 \text{ µsec}$$

Then the time for store and forward is

$$(\text{Switches} \times \text{Switch delay}) + ((\text{Switches} + 1) \times \text{Transfer time}) = (7 \times 0.25) + (8 \times 1) = 9.75 \text{ μsecs}$$

while wormhole routing is

$$(\text{Switches} \times \text{Switch delay}) + \text{Transfer time} = (7 \times 0.25) + 1 = 2.75 \text{ μsecs}$$

For this example, wormhole routing improves latency by more than a factor of three. ■

A final routing issue is the order in which packets arrive. Some networks require that packets arrive in the order in which they are sent. The alternative removes this restriction, requiring software to reassemble the packets in proper order.

Congestion Control

One advantage of a circuit-switched network is that once a circuit is established, it ensures there is sufficient bandwidth to deliver all the information that can be sent along that circuit. Thus interconnection bandwidth is reserved as circuits are established rather than consumed as data are sent, and if the network is full, no more circuits can be established. You may have encountered this blockage when trying to place a long distance phone call on a popular holiday, as the telephone system tells you that "all circuits are busy" and asks you to please call back at a later time.

Packet-switched networks do not reserve interconnect bandwidth in advance, so the interconnection network can become clogged with too many packets. Just as with rush hour traffic, a traffic jam of packets increases packet latency. Packets take longer to arrive, and in extreme cases fewer packets per second are delivered by the interconnect, just as is the case for the poor rush-hour commuters. There is even the computer equivalent of gridlock: *deadlock* is achieved when packets in the interconnect can make no forward progress no matter what sequence of events happens. Chapter 8 addresses how to avoid this ultimate congestion.

These problems are exacerbated with higher bandwidth and longer distance networks, as this Example illustrates.

EXAMPLE Assume a 155 Mbits/sec network stretching from San Francisco to New York City. How many bytes will be in flight? What is the number if the network is upgraded to 1000 Mbits/sec?

ANSWER The speed of light is still 299,792.5 kilometers per second, signals go at about 50% of the speed of light in a conductor, and the distance between San Francisco and New York City is 4120 km. Calculating time of flight:

$$\text{Time of flight} = \frac{4120 \text{ km}}{0.5 \times 299{,}792.5 \text{ km/sec}} = 0.0275 \text{ secs}$$

Let's assume the network delivers 50% of the peak bandwidth. The number of bytes in transit on a 155 Mbits/sec network is

$$\begin{aligned}\text{Bytes in transit} &= \text{Delivered bandwidth} \times \text{Time of Flight} \\ &= \frac{0.5 \times 155 \text{ Mbits/sec}}{8} \times 0.0275 \text{ secs} = 9.7 \text{ MB/sec} \times 0.0275 \text{ secs} \\ &= 260 \text{ KB}\end{aligned}$$

At 1000 Mbits/sec the number is

$$\begin{aligned}\text{Bytes in transit} &= \frac{0.5 \times 1000 \text{ Mbits/sec}}{8} \times 0.0275 \text{ secs} = 62.5 \text{ MB/sec} \times 0.0275 \text{ secs} \\ &= 1678 \text{ KB}\end{aligned}$$

Clearly a megabyte of messages will be a challenge to control and to store. ■

The solution to congestion is to prevent new packets from entering the network until traffic is reduced. Using our automobile analogy, this is the role of the metering lights on freeway on-ramps that control the rate of cars entering the freeway. There are three basic schemes used for congestion control in computer interconnection networks, each with its own weaknesses: packet discarding, flow control, and choke packets.

The simplest, and most callous, is *packet discarding*. If a packet arrives at a switch and there is no room in the buffer, the packet is discarded. This scheme relies on higher-level software that handles errors in transmission to resend lost packets. Internetworking protocols such as UDP discard packets.

The second scheme is to rely on *flow control* between pairs of receivers and senders. The idea is to use feedback to tell the sender when it is allowed to send the next packet. One version of feedback is via separate wires between adjacent senders and receivers that tell the sender to stop immediately when the receiver cannot accept another message. This *back-pressure* feedback is rapidly sent back to the original sender over dedicated lines, causing all links between the two end points to be frozen until the receiver can make room for the next message. Back-pressure flow control is common in MPPs. A more sophisticated variation of feedback is for the ultimate destination to give the original sender the right to send n packets before getting permission to send more. The collection of n packets is typically called a *window*, with the window's size determining the minimum frequency of communication from receiver to sender. The goal of the window is to send enough packets to overlap the latency of the interconnection with the overhead to send and receive a packet. The TCP protocol uses a window.

This brings us to a point of confusion on terminology in many papers and textbooks. Note that flow control describes just two nodes of the interconnection and not the total interconnection network between all end systems. *Congestion control* refers to schemes that reduce traffic when the collective traffic of all nodes is too large for the network to handle. Hence flow control helps congestion control, but it is not a universal solution.

The third scheme is based on *choke packets*. The observation is that you only want to limit traffic when the network is congested. The idea is for each switch to see how busy it is, entering a warning state when it passes a threshold. Each packet received by the switch in a warning state will be sent back to the source via a choke packet that includes the intended destination. The source is expected to reduce traffic to that destination by a fixed percentage. Since it likely will have already sent many packets along that path, it waits for all the packets in transit to be returned before taking choke packets seriously.

7.6 Practical Issues for Commercial Interconnection Networks

There are two practical issues in addition to the technical issues described so far that are important considerations for some interconnection networks: standardization and fault tolerance.

Standardization

Standards are useful in many places in computer design, but with interconnection networks they are often critical. Advantages of successful standards include low cost and stability: the customer has many vendors to choose from, which both keeps price close to cost due to competition and makes the viability of the interconnection independent of the stability of a single company. Components designed to be used in a standard interconnection may also have a larger market, and this higher volume can lower the vendor's costs, further benefitting the customer. Finally, a standard allows many companies to build products with interfaces to the standard, so the customer does not have to wait for a single company to develop interfaces to all the products the customer might be interested in.

One drawback of standards is that it takes a long time for committees to agree on the definition of standards, which is a problem when technology is changing quickly. Another problem is *when* to standardize: on one hand, designers would like to have a standard before anything is built; on the other, it would be better if something is built before standardization to avoid legislating useless features or omitting important ones. When done too early, it is often done entirely by committee, which is somewhat like asking all of France to prepare a single dish of food. Standards can also suppress innovation, since the interfaces are fixed by the standard.

MPP interconnection networks are traditionally proprietary, while LANs and WANs use standards. WANs involve many types of companies and must connect to many brands of computers, so it is difficult to imagine a proprietary WAN ever being successful. The ubiquitous nature of the Ethernet shows the popularity of standards for LANs as well as WANs, and it seems unlikely that many customers would tie the viability of their LAN to the stability of a single company.

Since an MPP is really a single brand of computer from a single company, the customer is already betting on a single company, removing one of the main arguments for interconnect standards. Thus the few MPPs that used standard interconnections did so to take advantage of the lower cost of components developed for these standards.

Node Failure Tolerance

The second practical issue refers to whether or not the interconnection relies on all the nodes being operational in order for the interconnection to work properly. Since software failures are generally much more frequent than hardware failures, the question is whether a software crash on a single node can prevent the rest of the nodes from communicating.

Clearly WANs would be useless if they demanded that thousands of computers spread across a continent be continuously available, and so they all tolerate the failures of individual nodes. LANs connect dozens to hundreds of computers together, and again it would be impractical to require that no computer ever fail. All successful LANs normally survive node failures.

Although some MPPs have the ability to work around failed nodes and switches, it is not clear that MPP operating systems support this feature. The close cooperation of the software on an MPP during communication also makes it unlikely that the interconnection would be useful if a single node crashed.

EXAMPLE Figure 7.20 shows the number of failures of 58 workstations on a local area network for a period of just over one year. Suppose that one local area network is based on a network that requires all machines to be operational for the interconnection network to send data; if a node crashes, it cannot accept messages, so the interconnection becomes choked with data waiting to be delivered. An alternative is the traditional local area network, which can operate in the presence of node failures; the interconnection simply discards messages for a node that decides not to accept them. Assuming that you need to have both your workstation and the connecting LAN to get your work done, how much greater are your chances of being prevented from getting your work done using the failure-intolerant LAN versus traditional LANs? Assume the down time for a crash is less than 30 minutes. Calculate using the one-hour intervals from this figure.

ANSWER Assuming the numbers for Figure 7.20, the percentage of hours that you can't get your work done using the failure-intolerant network is

$$\frac{\text{Intervals with failures}}{\text{Total intervals}} = \frac{\text{Total intervals} - \text{Intervals no failures}}{\text{Total intervals}}$$

$$= \frac{8974 - 8605}{8974} = \frac{369}{8974} = 4.1\%$$

The percentage of hours that you can't get your work done using the traditional network is just the time your workstation has crashed. Assuming that these failures are equally distributed among workstations, the percentage is

$$\frac{\text{Failures/Machines}}{\text{Total intervals}} = \frac{654/58}{8974} = \frac{11.28}{8974} = 0.13\%$$

Hence you are more than 30 times more likely to be prevented from getting your work done with the failure-intolerant LAN than with the traditional LAN, according to the failure statistics in Figure 7.20. Stated alternatively, the person responsible for maintaining the LAN would receive a thirtyfold increase in phone calls from irate users! ■

One practical issue is tied to node failure tolerance: If the interconnection can survive a failure, can it also continue operation while a new node is added to the interconnection? If not, the interconnection must be disabled each time a new node is added. Disabling is impractical for both WANs and LANs.

Finally, we have been discussing the ability of the network to operate in the presence of failed nodes. Clearly as important to the happiness of the network administrator is the reliability of the network media and switches themselves, for their failure is certain to frustrate much of the user community.

Failed machines per time interval	One-hour intervals with number of failed machines in first column	Total failures per one-hour interval	One-day intervals with number of failed machines in first column	Total failures per one-day interval
0	8605	0	184	0
1	264	264	105	105
2	50	100	35	70
3	25	75	11	33
4	10	40	6	24
5	7	35	9	45
6	3	18	6	36
7	1	7	4	28
8	1	8	4	32
9	2	18	2	18
10	2	20		
11	1	11	2	22
12			1	12
17	1	17		
20	1	20		
21	1	21	1	21
31			1	31
38			1	38
58			1	58
Total	**8974**	**654**	**373**	**573**

FIGURE 7.20 Measurement of reboots of 58 DECstation 5000s running Ultrix over a 373-day period. These reboots are distributed into time intervals of one hour and one day. The first column is used to sort the intervals according to the number of machines that failed in that interval. The next two columns concern one-hour intervals, and the last two columns concern one-day intervals. The second and fourth columns show the number of intervals for each number of failed machines. The third and fifth columns are just the product of the number of failed machines and the number of intervals. For example, there were 50 occurrences of one-hour intervals with two failed machines, for a total of 100 failed machines, and there were 35 days with two failed machines, for a total of 70 failures. As we would expect, the number of failures per interval changes with the size of the interval. For example, the day with 31 failures might include one hour with 11 failures and one hour with 20 failures. The last row shows the total number of each column: the number of failures don't agree because multiple reboots of the same machine in the same interval do not result in separate entries. (These data were collected by Randy Wang of U.C. Berkeley.)

7.7 Examples of Interconnection Networks

To further understand these issues, this section explores examples and the solutions used in each context. Figure 7.21 lists several examples and Figure 7.22 shows the packet formats for three examples. Figure 7.23 shows where networks are connected on several systems and the level of processing available on the network interface card. We discuss a few networks in more detail.

The first example is the Ethernet: It has been extraordinarily successful with the 10 Mbits/sec standard proposed in 1978 used practically everywhere today. Many classes of computers include Ethernet as a standard interface. This packet-switched network uses carrier sensing with exponential backoff to arbitrate for the network, and has been codified as IEEE 802.3.

Given that computers are hundreds of times faster than they were in 1978 and the shared interconnection is no faster, engineers have invented temporary solutions until a faster interconnect can take Ethernet's place. One solution is to use multiple Ethernets to connect machines and to connect these smaller Ethernets with devices that can take traffic from one Ethernet and pass it on to another as needed. These devices allow individual Ethernets to operate in parallel, thereby increasing the aggregate interconnection bandwidth of a collection of computers. In effect these devices provide similar functionality to the switches described above for point-to-point networks.

Figure 7.24 shows the potential parallelism. Depending on how they pass traffic and what kinds of interconnections they can put together, these devices are named differently:

- *Bridges*—These devices connect LANs together, passing traffic from one side to another depending on the addresses in the packet. Bridges operate at the Ethernet protocol level and are usually simpler and cheaper than routers, discussed next.
- *Routers* or *gateways*—These devices connect LANs to WANs or WANs to WANs and resolve incompatible addressing. Generally slower than bridges, they operate at the internetworking protocol level (see section 7.9). Routers divide the interconnect into separate smaller subnets, which simplifies manageability and improves security.

Since these devices were not planned as part of the Ethernet standard, their ad hoc nature has added to the difficulty and cost of maintaining LANs.

	MPP				LAN				WAN
	CM-5	**IBM SP-2**	**Intel Paragon**	**Cray T3D**	**Ethernet**	**100-Mb Ethernet**	**Switched Ethernet**	**FDDI**	**ATM**
Length (meters)	25	10?	10?	10.3	500/ 2500	200	500/2500	4000	100/1000
Number data lines	4	8	16	16	1	1	1	1	1
Clock rate (MHz)	40	40	100	150	10	100	10	100	155/622…
Switch?	Yes	Yes	Yes	Yes	No	No	Yes	No	Yes
Nodes	≤2048	≤512	≤1024	≤2048	≤254	≤254	≤254	≤254	≈10000
Material	Copper	Copper	Copper	Copper	Copper	Copper	Copper	Fiber	Copper/fiber
Bisection BW (Mbits/sec)	40x Nodes	320x Nodes	1600x $Nodes^{1/2}$	2400x $Nodes^{2/3}$	10	100	10x Nodes	100	155x Nodes
Peak link BW (Mbits/sec)	160	320	1600	2400	10	100	10	100	155/622
Measured link BW	160	284	1400	1120	9			97	10
Latency (µsecs)	5	1	1	0.2	15	1.5	≈50	10	≈50
Send + receive overheads (µsecs)	15	39	24	0.7	440	440	440	≈500	630
Topology	Fat tree	Fat tree	2D mesh	3D torus	Line	Line	Star	Ring	Star
Connec-tionless?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Store & forward?	No	No	No	No	No	No	No	No	Yes
Congestion control	Back-pressure	Back-pressure	Back-pressure	Back-pressure	Carrier sense	Carrier sense	Carrier sense	Token	
Standard	No	No	No	No	IEEE 802.3				ATM Forum
Fault tolerance	No	Yes	No	No	Yes	Yes	Yes	Yes	Yes

FIGURE 7.21 Several examples of MPP, LAN, and WAN interconnection networks. The overhead figure is hardware and software overhead, measured on SPARCStation-10s for the LANs and WAN (see section 7.10).

One potential successor to Ethernet is FDDI, which stands for *fiber-distributed data interface.* This optical-based interconnection was specified at 100 Mbits/sec

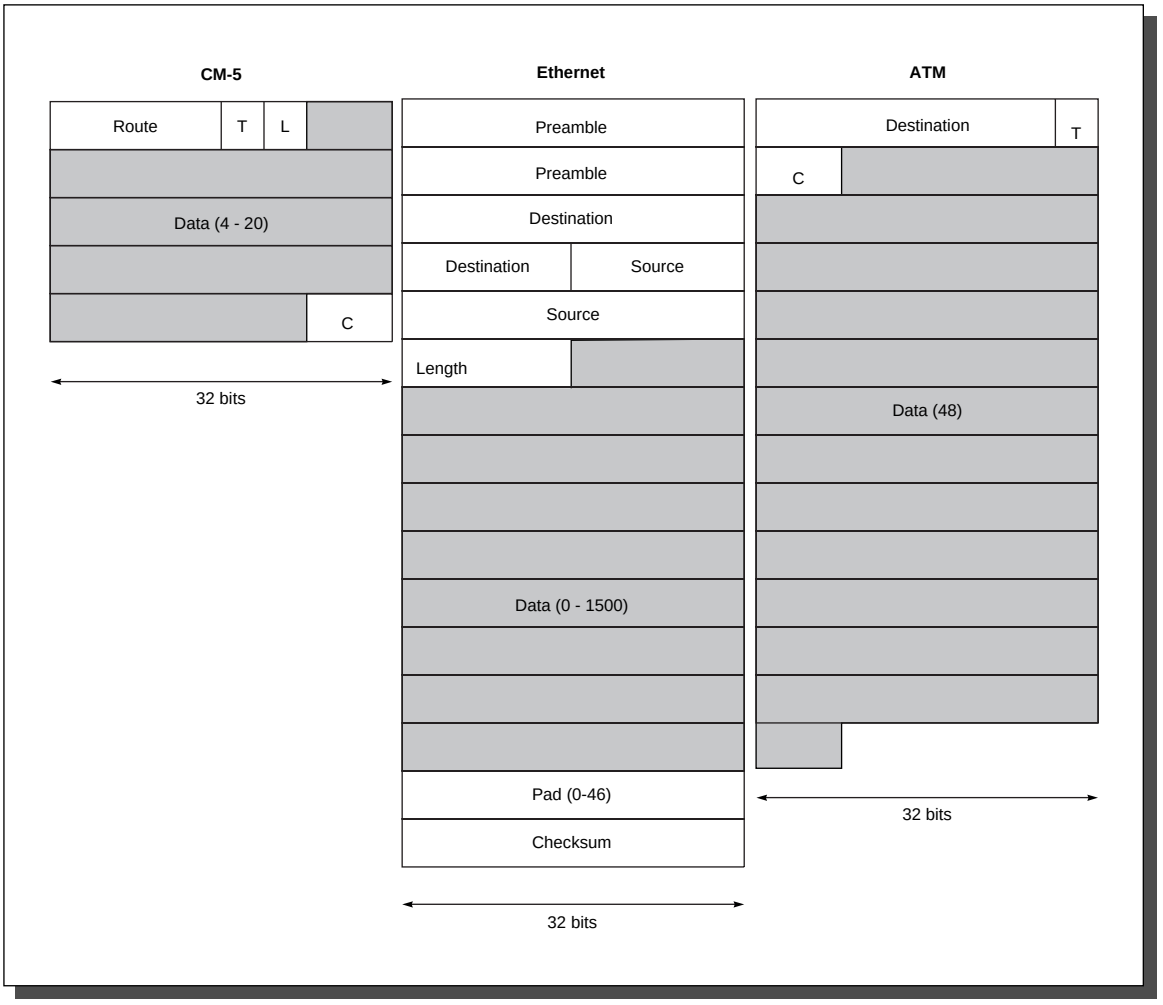

FIGURE 7.22 Packet format for CM-5, Ethernet, and ATM. ATM calls their messages "cells" instead of packets, so this is more properly called the ATM cell format (see section 7.10). The width of each drawing is 32 bits. All three formats have destination addressing fields, encoded differently for each situation. All three also have a checksum field (C) to catch transmission errors, although the ATM checksum field is calculated only over the header; ATM relies on higher-level protocols to catch errors in the data. Both CM-5 and the Ethernet have a length field (L), since the packets hold a variable amount of data, with the former counted in 32-bit words and the latter in bytes. The CM-5 and ATM headers have a type field (T) that gives the type of packet. The remaining Ethernet fields are a preamble to allow the receiver to recover the clock from the self-clocking code used on the Ethernet, the source address, and a pad field to make sure the smallest packet is 64 bytes (including the header).

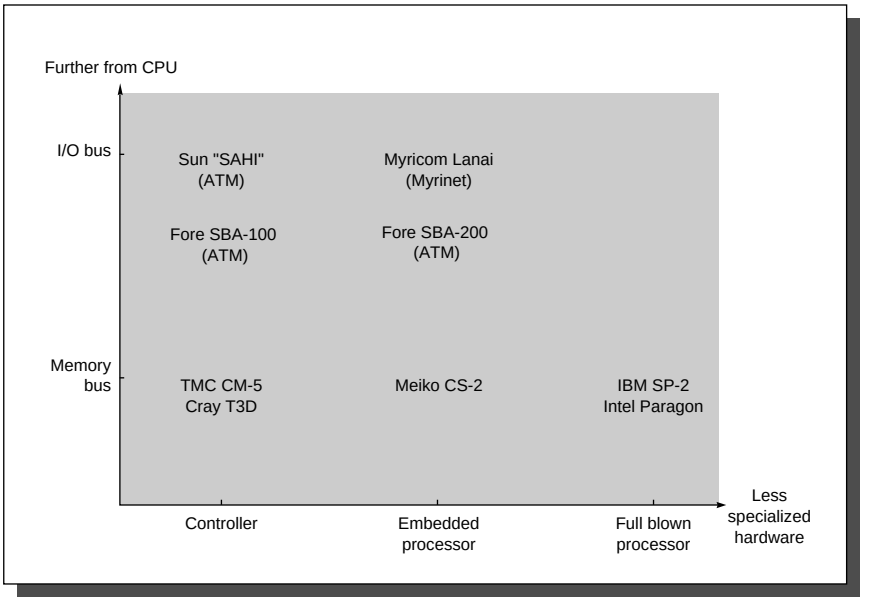

FIGURE 7.23 Location of network interface versus processing on network interface for several interconnection networks. The fallacy on page 623 refers to the increasing processing performance. (From a presentation by Lok Liu of U.C. Berkeley.)

and could use much longer cables than Ethernet. Unfortunately, FDDI inherited Ethernet's weakness in that all machines shared the interconnection medium, with only one packet on the medium at a time, so FDDI still needed bridges and routers. FDDI is currently most widely used as a "backbone" network, which simply means connecting LANs together via their routers rather than connecting desktop computers.

Today, there are three more promising LAN candidates. The first candidate is one of two competing standards that offer a 100 Mbits/sec version of the Ethernet. The second builds on the trend toward many smaller networks by making switches a part of the standard. *Switched Ethernet* simply includes fast, multiport switches so that the bandwidth to a single machine is no higher, but the aggregate bandwidth of the LAN is much higher. Twisted pairs can connect the machine to the switch, lowering the costs of wiring. The third candidate, *asynchronous transfer mode* (ATM), is described in section 7.10.

At the opposite performance end of the LAN networks is the MPP network found in the Cray Research T3D. Using 16-bit links clocked at 150 MHz yields 2400 megabits (300 MB) per second per link. The distributed switch using a

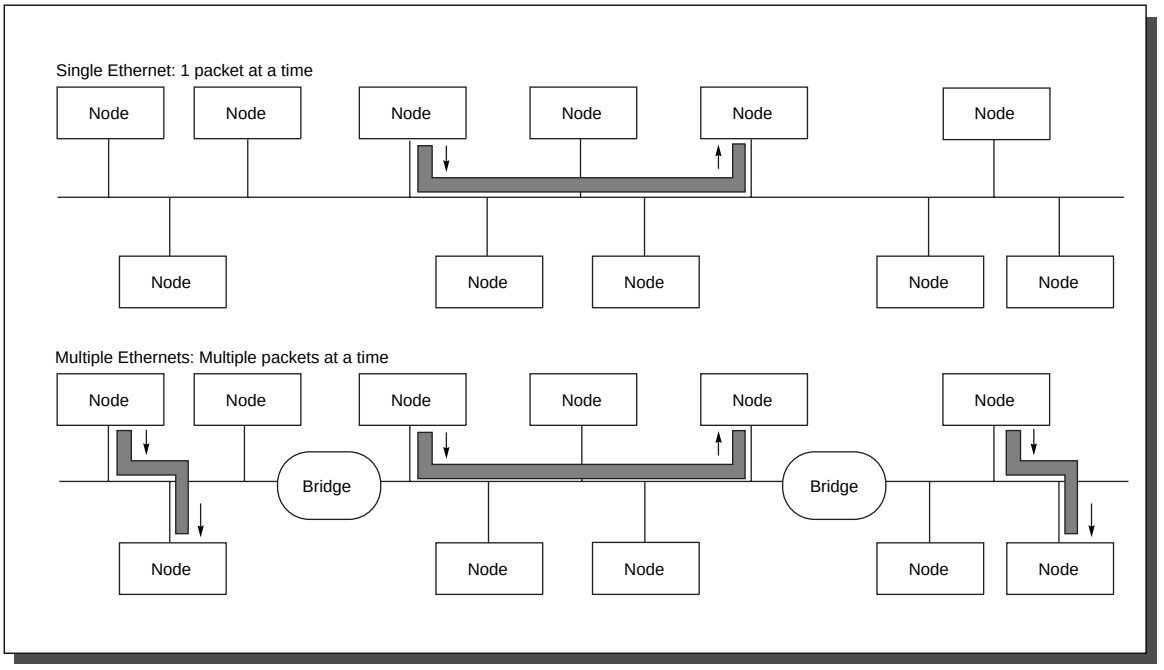

FIGURE 7.24 The potential increased bandwidth of using many Ethernets and bridges.

three-dimensional torus topology has a bisection bandwidth of between about 10% and 25% of a fully connected system—38,400 to 244,000 Mbits/second—depending on the number of nodes. This proprietary network, which can scale up to 2048 nodes, has a one-way trip latency of less than one microsecond, including hardware and software overhead. Although the T3D network does not face the LAN challenges of distance and security, it still shows that LANs have a long way to go before pushing the envelope of networking bandwidth and latency.

7.8 Crosscutting Issues for Interconnection Networks

This section describes four topics discussed in other chapters that are fundamental to interconnections.

Efficient Interface to Memory Hierarchy versus Interconnection Network

Traditional evaluations of processor performance, such as SPECint and SPECfp, encourage the memory hierarchy to be closely integrated with the processor, as

the efficiency of the memory hierarchy translates directly into processor performance. Hence microprocessors have first-level caches on chips along with buffers for writes, and usually have second-level caches immediately next to the chip. Benchmarks such as SPECint and SPECfp do not reward good interfaces to interconnection networks, and hence many machines make the access time to the network delayed by the full memory hierarchy: Writes must lumber their way through full write buffers, and reads must go through the cycles of first- and second-level cache misses before reaching the interconnection. This results in newer systems having higher latencies to interconnections than older machines.

Let's compare three machines. A 40-MHz SPARCstation-2, a 50-MHz SPARCstation-20 without an external cache, and a 50-MHz SPARCstation-20 with an external cache. According to SPECint95, this list is in order of increasing performance. The time to access the I/O bus (S-bus), however, increases in this sequence: 200 ns, 500 ns, and 1000 ns. The SPARCstation-2 is fastest because it has a single bus for memory and I/O, and there is only one level to the cache. The SPARCstation-20 memory access must first go over the memory bus (M-bus) and then to the I/O bus, adding 300 ns. Machines with a second-level cache pay an extra penalty of 500 ns before accessing the I/O bus.

Compute-Optimized Processors versus Receiver Overhead

The overhead to receive a message likely involves an interrupt, which bears the cost of flushing and then restarting the processor pipeline. As mentioned earlier, to read the network status and to receive the data from the network interface likely operates at cache miss speeds. As microprocessors become more superscalar and go to faster clock rates, the number of missed instruction issue opportunities per message reception seems likely to rise quickly over time.

Where to Draw the Hardware/Software Dividing Line for Interconnection Network Functions

The choice of hardware versus software support for interconnection networks has many of the same trade-offs as in other parts of a computer. Examples of features implemented in hardware with some systems and in software with others are message routing, message error detection, and retransmission.

Early MPPs required each processor along a path to route a message from switch to switch. This software overhead dominated switch latency, and thus all recent MPPs have hardware routing. Hardware routing in turn affects the complexity of the topologies of the interconnection, since hardware simplicity demands a simple routing algorithm.

Another example of the hardware/software implementation decision is reliable delivery of messages. TCP checks to make sure that messages are delivered reliably by including checksums, yet LANs like Ethernet will also include checksums in hardware to determine whether packets have been reliably delivered. Some interconnection interfaces hold onto a packet until it has been acknowledged by the receiving hardware, retransmitting in case of failure. These same functions are provided in software by TCP. Reliable delivery is one area of considerable duplication in effort between the hardware and software rather than a clear division of responsibilities (see the pitfall on page 623).

Protection and User Access to the Network

The challenge is to ensure safe communication across a network without invoking the operating system in the common case. The Cray Research T3D offers an interesting case study. It supports a global address space, so loads and stores can access memory across the network. Protection is ensured because each access is checked by the TLB.

To support transfer of larger objects, a block transfer engine (BLT) was added to the hardware. Protection of access requires invoking the operating system, before using the BLT, to check the range of accesses to be sure there will be no protection violations.

Figure 7.25 compares the bandwidth delivered as the size of the object varies for reads and writes. For very large reads, 512 KB, the BLT does achieve the highest performance: 140 MB/sec. But simple loads get higher performance for 8 KB or less. For the write case, both achieve a peak of 90 MB/sec, presumably because of the limitations of the memory bus. But for writes, BLT can only match the performance of simple stores for transfers of 2 MB; anything smaller and it's faster to send stores. Clearly a BLT that avoided invoking the operating system in the common case would be more useful.

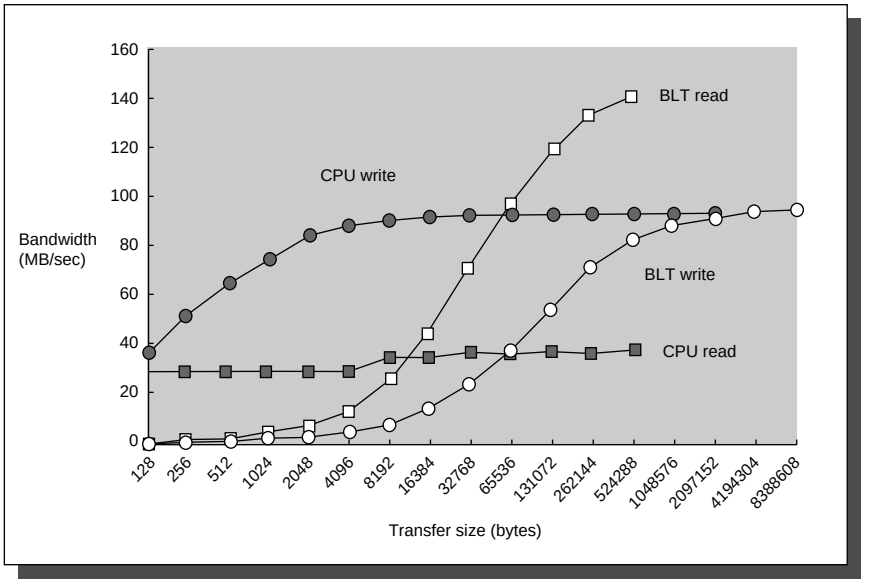

FIGURE 7.25 Bandwidth versus transfer size for simple memory access instructions versus a block transfer device on the Cray Research T3D. (Arpaci et al. [1995].)

7.9 Internetworking

Undoubtedly one of the most important innovations in the communications community has been internetworking. It allows computers on independent and incompatible networks to communicate reliably and efficiently. The need to cross networks is illustrated by Figure 7.26, which shows the networks and machines involved in transferring a file from Stanford University to the University of California at Berkeley, a distance of about 75 km.

The low cost of internetworking is remarkable. For example, it is vastly less expensive to send electronic mail than to make a coast-to-coast telephone call and leave a message on an answering machine. This dramatic cost improvement is achieved using the same long-haul communication lines as the telephone call, which makes the improvement even more impressive.

The enabling technologies for internetworking are software standards that allow reliable communication without demanding reliable networks. The underlying principle of these successful standards is that they were composed as a hierarchy of layers, each layer taking responsibility for a portion of the overall communication task. Each computer, network, and switch implements its layer of the standards, relying on the other components to faithfully fulfill their responsibilities. These layered software standards are called *protocol families* or *protocol*

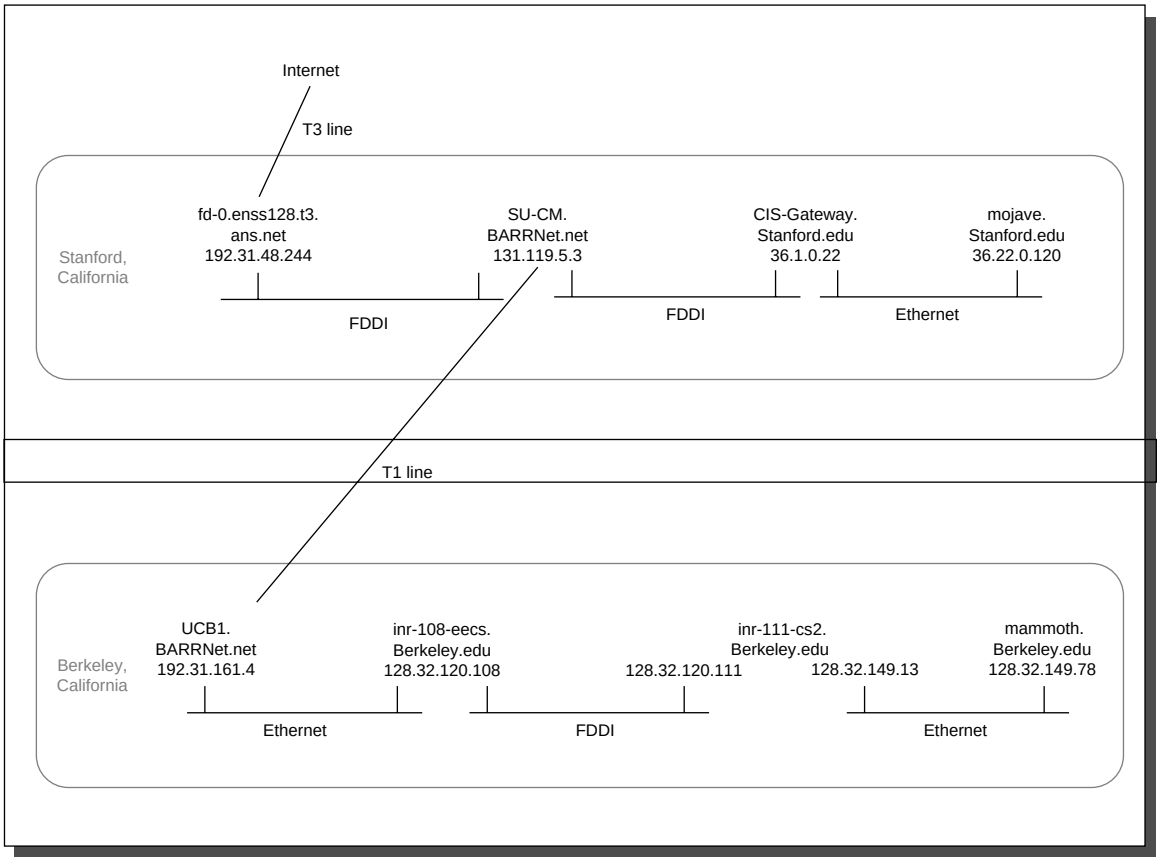

FIGURE 7.26 The connection established between mojave.stanford.edu and mammoth.berkeley.edu. FDDI is a 100 Mbits/sec LAN, while a T1 line is a 1.5 Mbits/sec telecommunications line and a T3 is a 45 Mbits/sec telecommunications line. BARRNet stands for Bay Area Research Network. Note that inr-111-cs2.Berkeley.edu is a router with two Internet addresses, one for each port.

suites. They enable applications to work with any interconnection without extra work by the application programmer. Figure 7.27 suggests the hierarchical model of communication.

The most popular internetworking standard is *TCP/IP*, which stands for *transmission control protocol*/*internet protocol*. This protocol family is the basis of the humbly named *Internet*, which connects tens of millions of computers around the world. This popularity means TCP/IP is used even when communicating locally across compatible networks; for example, the network file system NFS uses IP even though it is very likely to be communicating across a homogenous LAN such as Ethernet.

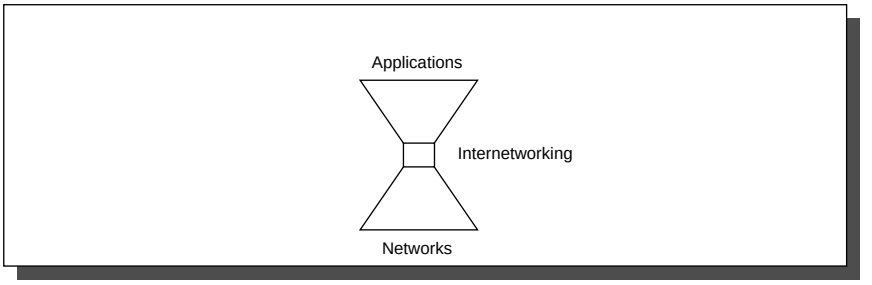

FIGURE 7.27 The role of internetworking. The width is intended to indicate the relative number of items at each level.

We use TCP/IP as our protocol family example; other protocol families follow similar lines. Section 7.13 gives the history of TCP/IP.

The goal of a family of protocols is to simplify the standard by dividing responsibilities hierarchically among layers, with each layer offering services needed by the layer above. The application program is at the top, and at the bottom is the physical communication medium, which sends the bits. Just as abstract data types simplify the programmer's task by shielding the programmer from details of the implementation of the data type, this layered strategy makes the standard easier to understand.

The key to protocol families is that communication occurs *logically at the same level* of the protocol in both sender and receiver, but it is *implemented via services of the lower level*. This style of communication is called *peer-to-peer*. As an analogy, imagine that General A needs to send a message to General B on the battlefield. General A writes the message, puts it in an envelope addressed to General B, and gives it to a colonel with orders to deliver it. This colonel puts it in an envelope and writes the name of the corresponding colonel who reports to General B, and gives it to a major with instructions for delivery. The major does the same thing and gives it to a captain, who gives it to a lieutenant, who gives it to a sergeant. The sergeant takes the envelope from the lieutenant, puts it into an envelope with the name of a sergeant who is in General B's division, and finds a private with orders to take the large envelope. The private borrows a motorcycle and delivers the envelope to the sergeant. Once it arrives, it is passed up the chain of command, with each person removing an outer envelope with his name on it and passing on the inner envelope to his superior. As far as General B can tell, the note is from another general. Neither general knows who was involved in transmitting the envelope, nor how it was transported from one division to the other.

Protocol families follow this analogy more closely than you might think, as Figure 7.28 shows. The original message is given a header and possibly a trailer to be sent by the lower-level protocol. The next-lower protocol in turn adds its own header to the message, possibly breaking it up into smaller messages if it is

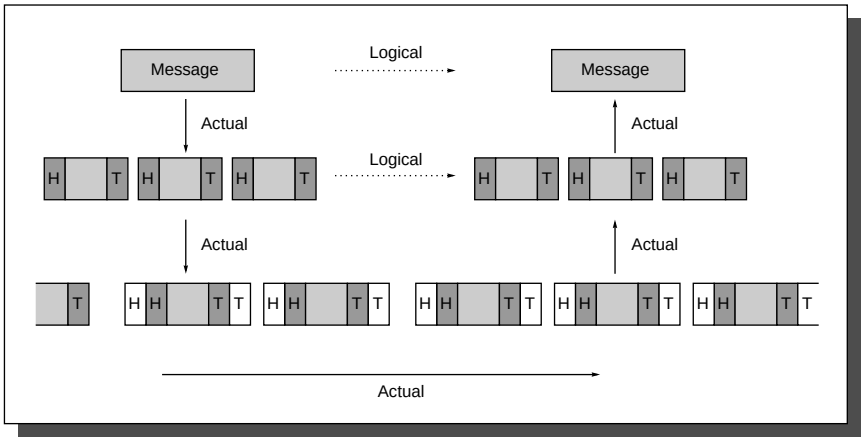

FIGURE 7.28 A generic protocol stack with two layers. Note that communication is peer-to-peer, with headers and trailers for the peer added at each sending layer and removed by each receiving layer. Each layer is intended to offer services to the one above to shield it from unnecessary details.

too large for this layer. Recalling our analogy, a long message from the general would be divided and placed in several envelopes if it could not fit in one. This division of the message and appending of headers and trailers continues until the message descends to the physical transmission medium. The message is then sent to the destination. Each level of the protocol family on the receiving end will check the message at its level and peel off its headers and trailers, passing it on to the next higher level and putting the pieces back together. This nesting of protocol layers for a specific message is often referred to as a *protocol stack*, reflecting the last-in-first-out nature of the addition and removal of headers.

As in our analogy, the danger in this layered approach is the considerable latency added to message delivery. Clearly one way to reduce latency is to reduce the number of layers. But keep in mind that protocol families are used to define a standard, not to force how the standard is implemented. Just as there are many ways to implement an instruction set architecture, there are many ways to implement a protocol family.

Our protocol stack example is TCP/IP. Let's assume that the bottom protocol layer is Ethernet. The next level up is the Internet Protocol or IP layer; the official term for an IP packet is *datagram*. The IP layer routes the datagram to the destination machine, which may involve many intermediate machines or switches. IP makes a best effort to deliver the packets, but does not guarantee delivery, content, or order of datagrams. The TCP layer above IP makes the guarantee of reliable, in-order delivery and prevents corruption of datagrams.

Following the example in Figure 7.28, assume an application program wants to send a message to a machine via an Ethernet. It starts with TCP. The largest

number of bytes that can be sent at one time with TCP is 64 KB. Since the data may be much larger than 64 KB, TCP must divide it into smaller segments and reassemble them in proper order upon arrival. TCP adds a 20-byte header (Figure 7.29) to every datagram, and passes them down to IP. The IP layer above the physical layer adds a 20-byte header, also shown in Figure 7.29. The data sent down from the IP level to the Ethernet would be sent in packets with the format shown in Figure 7.22 on page 603. Note that the TCP packet appears inside the data portion of the IP datagram, just as Figure 7.28 suggests.

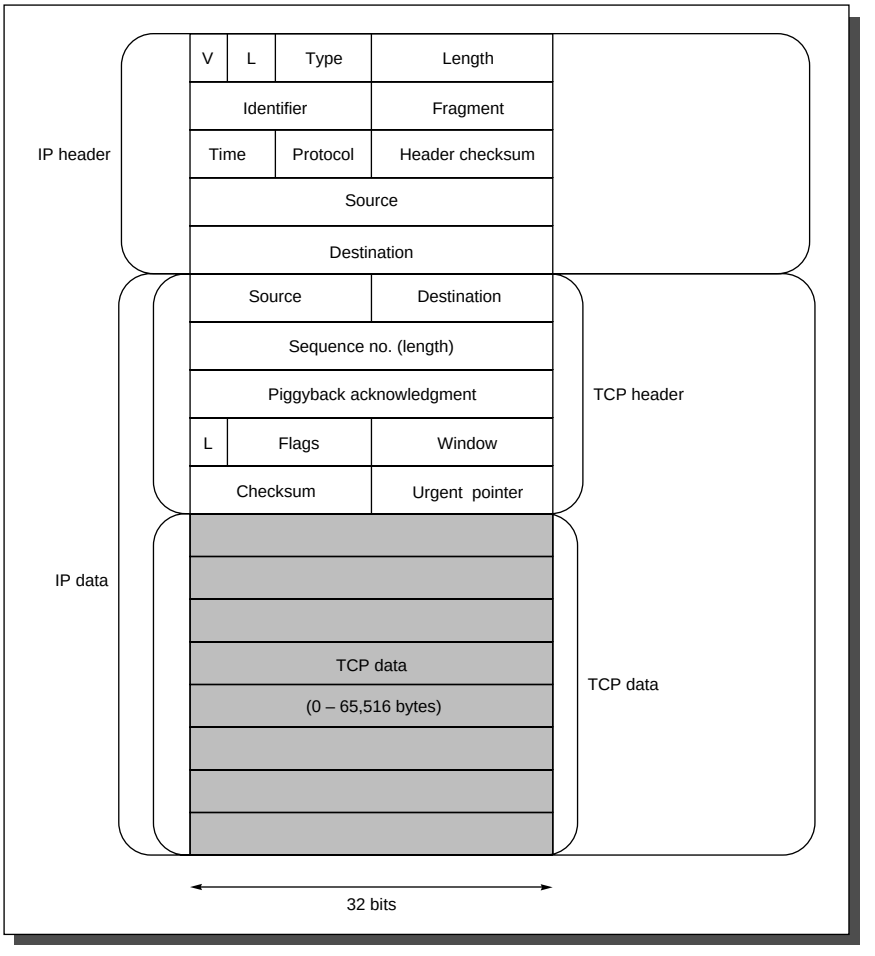

FIGURE 7.29 The headers for IP and TCP. This drawing is 32 bits wide. The standard headers for both are 20 bytes, but both allow the headers to optionally be expanded for rarely transmitted information. Both headers have a length of header field (L) to accommodate the optional fields, as well as source and destination fields. The length field of the whole datagram is in a separate length field in IP, while TCP combines the length of the datagram with

the sequence number of the datagram by giving the sequence number in bytes. TCP uses the checksum field to be sure that the datagram has not been corrupted, and the sequence number field to be sure the datagrams are assembled into the proper order when they arrive. IP provides checksum error detection only for the header, since TCP has protected the rest of the packet. One optimization is that TCP is allowed to send a sequence of datagrams before waiting for permission to send more. The number of datagrams that can be sent without waiting for approval is called the *window*, and the window field tells how many bytes may be sent beyond the byte being acknowledged by this datagram. TCP will adjust the size of the window depending on the success of the IP layer in sending datagrams; the more reliable and faster it is, the larger TCP makes the window. Since the window slides forward as the data arrives and is acknowledged, this technique is called a *sliding window protocol*. The *piggyback acknowledgment field* of TCP is another optimization. Since some applications send data back and forth over the same connection, it seems wasteful to send a datagram containing only an acknowledgment. This piggyback field allows a datagram carrying data to also carry the acknowledgment for a previous transmission, "piggybacking" on top of a data transmission. The *urgent pointer field* of TCP gives the address within the datagram of an important byte, such as a break character. This pointer allows the application software to skip over data so that the user doesn't have to wait for all prior data to be processed before seeing a character that tells the software to stop. The *identifier field* and *fragment field* of IP allow intermediary machines to break the original datagram into many smaller datagrams. A unique identifier is associated with the original datagram and placed in every fragment, with the fragment field saying which piece is which. The *time-to-live field* allows a datagram to be killed off after going through a maximum number of intermediate switches no matter where it is in the network. Knowing the maximum number of hops that it will take for a datagram to arrive—if it ever arrives—simplifies the protocol software. The *protocol field* identifies which possible upper layer protocol sent the IP datagram; in our case it is TCP. The V (for *version*) and *type fields* allow different versions of the IP protocol software to be used in the network. Explicit version numbering is included so that software can be upgraded gracefully machine by machine, without shutting down the entire network.

7.10 Putting It All Together: An ATM Network of Workstations

The search for a successor to the Ethernet has even attracted the attention of the telecommunications industry. Given the desirability of using switches and the need to efficiently interface to WANs, why not use ATM as a LAN? In addition to having the scalable bandwidth of switched Ethernet, ATM is defined independently from the physical medium, which allows the system to be upgraded gracefully to higher-speed interconnections. Unlike Ethernet, several data rates are included in the standard. There is also the practical hope that the volume of ATM equipment needed by the telecommunications industry will result in lower costs for LAN applications.

ATM is still evolving as a standard, so we describe the subset of ATM that today seems likely to be popular for LAN. (See section 7.13 for more about ATM and references.) The promise of ATM is that it will allow much higher performance communication, while avoiding the hodgepodge of devices needed to connect segments of LANs together or to connect LANs to WANs, since the same underlying technology can be used from top to bottom.

Our example is 16 SPARCstation-10 workstations connected via two brands of ATM switches. Figure 7.30 shows details of the organization. The SPARCstation-10 uses a 50-MHz SuperSPARC microprocessor, which consists of a

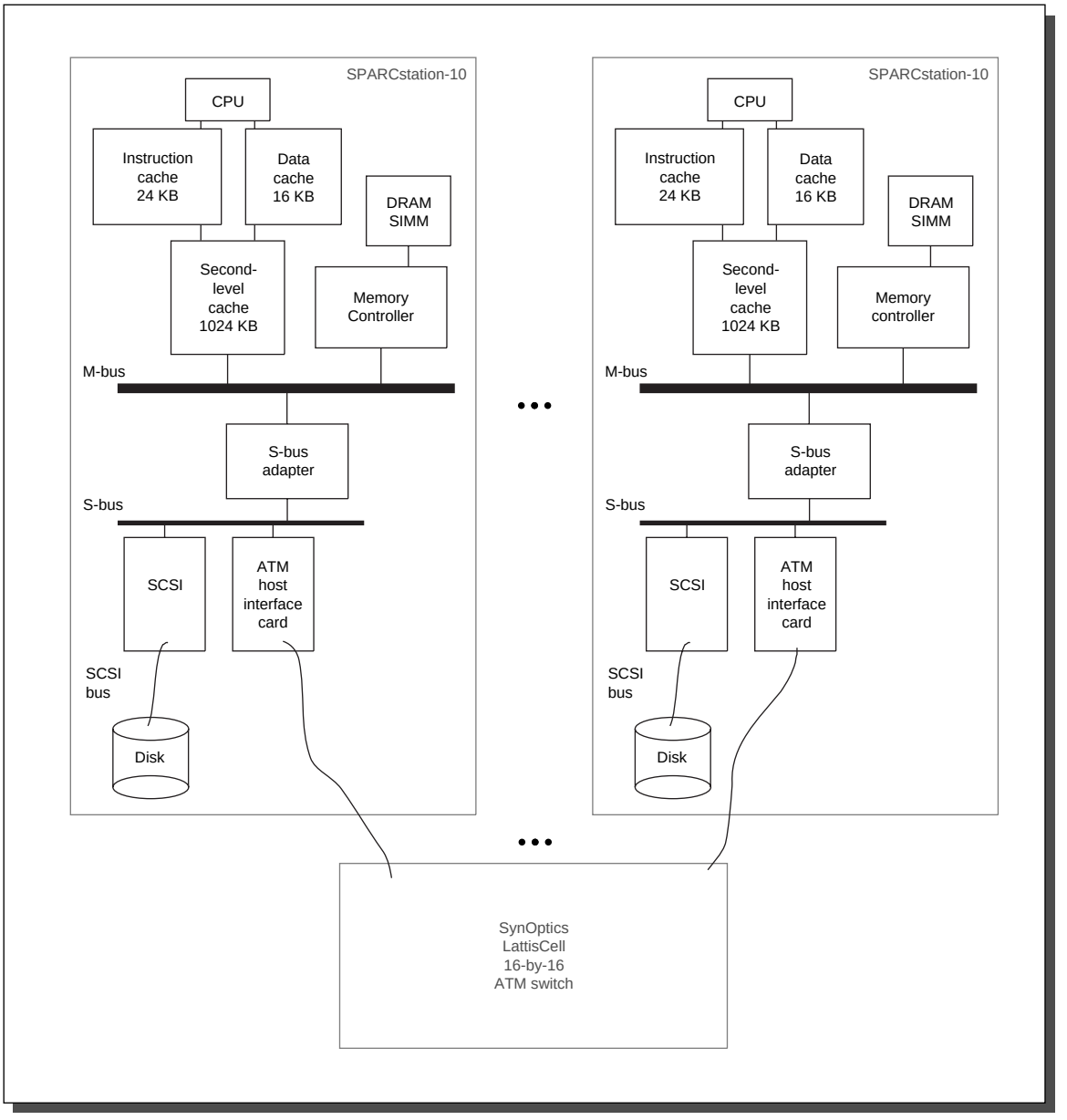

FIGURE 7.30 The organization of the SPARCstation-10 and the ATM switch. The switch is described in Figure 7.35.

superscalar CPU and instruction and data caches on a single chip. This chip interfaces to a 1-MB second-level cache that stands between the microprocessor and the 40-MHz M-bus. The M-bus connects to memory and to a bus interface to the 20-MHz S-bus, Sun's I/O bus. It is on the S-bus that we find the ATM *host interface card*, which interfaces to the 16-by-16 ATM switch.

Let's follow a transfer through the levels before we talk in detail about the components. The transfer proceeds down the TCP and IP layers until it reaches ATM. At the top of the ATM software is the ATM adaptation layer. Figure 7.31 shows this relationship between layers and Figure 7.32 shows AAL-5, the fifth proposal for an ATM standard protocol.

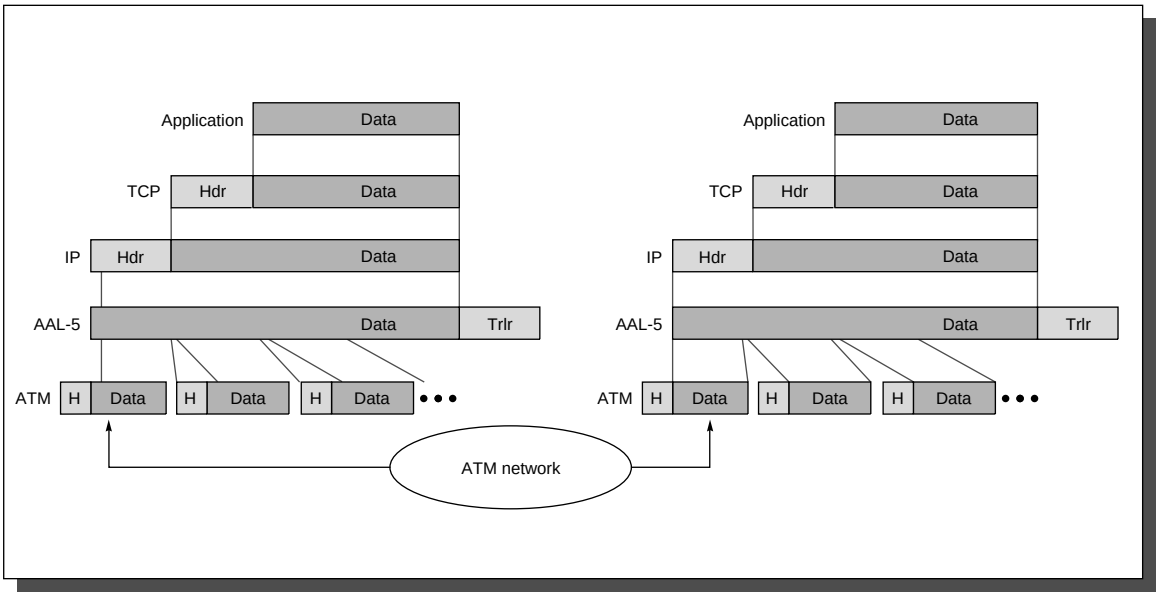

FIGURE 7.31 The relationship between TPC, IP, and AAL-5 layers. The ATM Forum hopes that over time adaptation layers such as AAL-5 will replace protocol suites such as TCP/IP.

The next layer of ATM software breaks up the data payload into the 48-byte units transferred in each ATM cell. Figure 7.33 shows the details of the ATM header. Unlike Ethernet, ATM is based on connections set up in advance before data are transmitted between two machines. For this example, we assume that the connections have already been established between machines, so that we do not need to establish a connection. These connections are called *virtual channels*, with the *virtual channel identifier* (VCI) used to pick the proper connection for each ATM cell. The ATM switch uses the VCI to send the cell to the proper output port. The header also provides a *virtual path identifier* (VPI); *virtual paths*

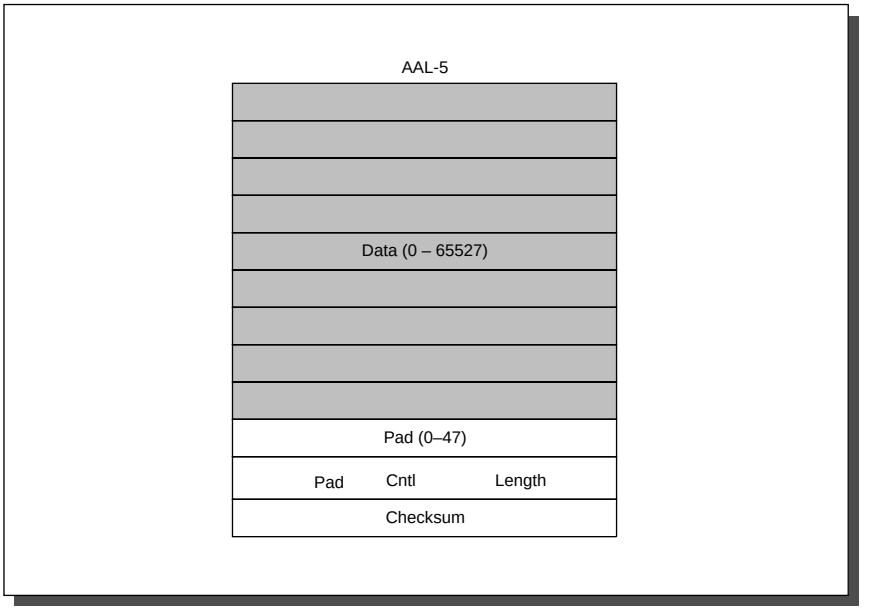

FIGURE 7.32 The AAL-5 format of ATM. The evolving ATM standard has at least five proposals for AALs, but in our view AAL-5 is the most likely winner for LANs when shipping datagrams. Lower-level protocols set bits in packets to show the end of the transfer. The pad field ensures that the data payload plus the AAL-5 trailer is a multiple of 48 bytes, the data size of an ATM cell. The control field (Cntl) is reserved for future use. Length and checksum fields are self-explanatory. The total size including trailer and pad is 64 KB.

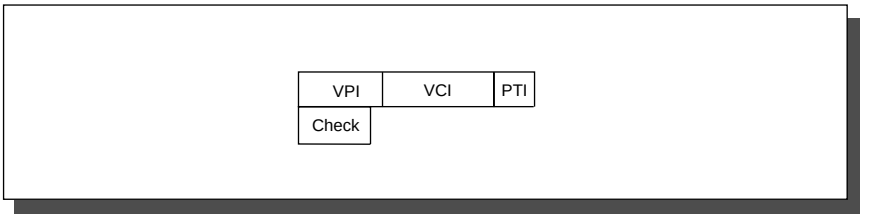

FIGURE 7.33 The ATM header format. The VPI field is 12 bits, VCI is 16 bits, the PTI field is 4 bits, and the checksum field is 8 bits. The main use of the payload type indicator field is to mark the end of the AAL-5 packet. This format is used for the ATM network-to-network interface.

simply represent bundles of virtual channels and may be used by ATM switches to route many channels at once rather than individually. The payload type indicator field is used by AAL-5 to mark the end of a packet.

Note that ATM uses a fixed-sized unit of transmission to simplify the design of switches and interfaces. These units are called *cells*, since use of the term *packet* in a connection-based system might lead to confusion. The cell selects the *connection* rather than including the *destination address* in every cell.

Once all cells composing the AAL-5 message are delivered to the appropriate machine, the cells arrive in order and are passed to the AAL-5 layer software. The layer checks the data and passes them up to the IP software, proceeding up the protocol stack.

The Host Interface Card

The ATM *host interface card* lives on the S-bus and interfaces a node to the ATM switch. The Fore Systems SBA-200 ATM adapter is a rather sophisticated network interface. It is the successor to Fore's simple SBA-100 adapter, which contained only send and receive FIFOs. The three key features of the SBA-200 are a 25-MHz Intel i960CA used as communications coprocessor, a DMA interface to the host bus, and an AAL5-compatible hardware CRC generator/checker. In addition, the SBA-200 has 256 KB of static RAM, which serves as program and data memory for the i960 and can be accessed directly from the host workstation processor. Figure 7.34 shows a block diagram of the SBA-200.

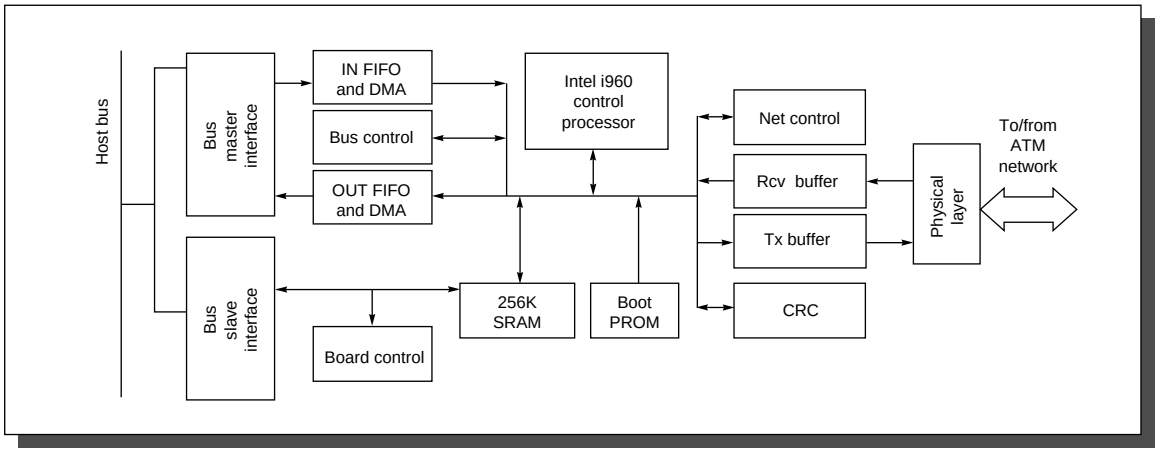

FIGURE 7.34 The Fore Systems SBA-200 host interface card. The processor and SRAM are in the center, the DMA unit in the upper-left corner, the direct access to the SRAM and to a few board control registers in the lower-left corner, and a simple send/receive FIFO interface to the fiber augmented with the CRC generator/checker on the right-hand side.

The inclusion of a processor on the network interface card is the most visible feature of the SBA-200. The main role of the i960 is to free the host workstation CPU from performing the ATM adaptation layer (AAL) segmentation and

reassembly of data payload into cells. The host CPU hands a data descriptor to the i960, which breaks the data blocks into cells and generates the cell headers and trailers, including appropriate checksums, or CRCs. On the receiving side, the i960 reassembles arriving cells into data blocks and hands a descriptor to the host once the data are complete.

The role of the DMA interface is twofold: It allows the ATM adaptor to transfer packets to and from main memory without host processor intervention, and it allows data to be transferred using burst bus transactions. The SBA-200's DMA interface can transfer data at over 30 Mbytes/sec in eight-word bursts.

The software support for the SBA-200 consists of two major components: the firmware that is downloaded into the SBA-200's SRAM at boot time and controls the operation of the i960, and the device driver that communicates with the i960 from the host. The apparent rationale underlying the design of Fore's hardware and software is to off-load the specifics of the ATM adaptation layer processing from the host processor as much as possible and to provide a data block interface to the device driver.

The Switches

Each host interface card is connected to a 16-by-16 ATM switch. Depending on the distance to the switch, the media can be either twisted pair copper wire or optical fiber. The latter option requires expensive electrical-to-optical interfaces, but stretches the distance between the host interface card and switch from meters to kilometers.

The Bay Networks LattisCell 10114 switch itself is a multistage switch similar to the Omega switch in Figure 7.13 (page 584), with an internal bandwidth that is twice the demand of the sixteen 155 Mbits/sec ports: 5 Gbits/sec. Figure 7.35 gives details of the switch.

Another ATM switch is the Fore Systems ASX-200. Inside this 2.5 Gbits/sec switch is simply a 40-MHz, 64-bit-wide bus! Capable of handling up to 24 155 Mbits/sec ports, the latency through the switch is less than 10 microseconds. It includes a SPARC microprocessor for routing and up to 700 KB of output buffer per port.

Performance of SPARCstations and ATM

To put ATM in context, this section compares the performance of two of the initial ATM switches to an Ethernet LAN. For each experiment, two Sparc 10s running Solaris 2.3 were connected to an ATM switch via OC-3C SONET links, which have a line rate of 155 Mbits/sec. The network interface was the Fore Systems SBA-200 ATM interface and two ATM switches were tested, the Fore Systems ASX-200 and the Bay Networks LattisCell 10114. The approximate 1995 prices of the components are shown in Figure 7.36.

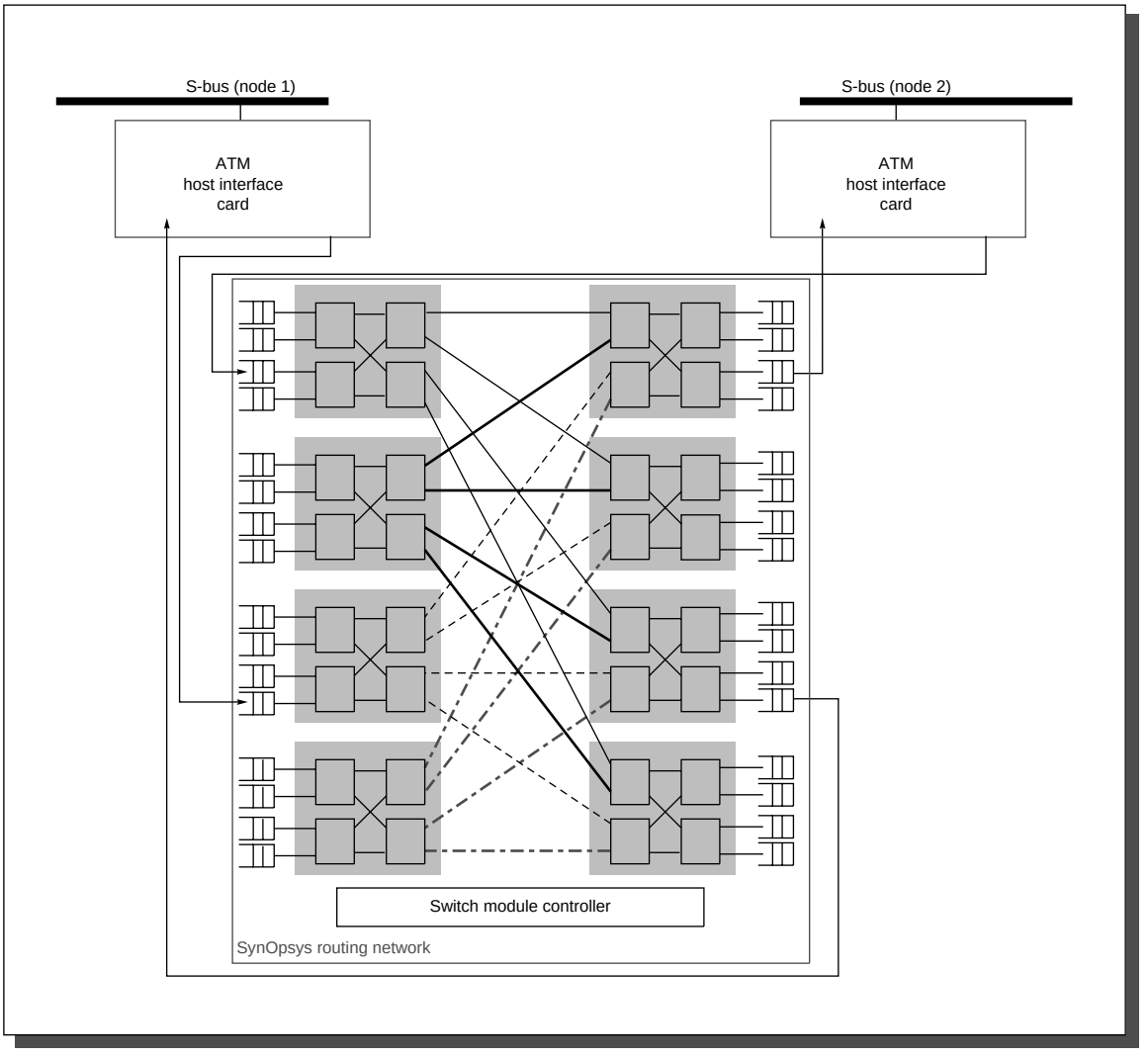

FIGURE 7.35 ATM host interface cards and 16-by-16 LattisCell ATM switch from Synoptics. The switch connection is called a Delta connection, which is a relative of the Omega switch. The switch also has a copy network that is identical to the routing network to support multicast; it is placed before the router. The switch module finds the outgoing destination port of the switch by indexing into a table using the incoming VCI port number.

Figure 7.37 shows the TCP throughput as the size of the TCP packets are increased for Ethernet and ATM. Sending larger packets amortizes the overhead of TCP, IP, and the ATM driver. Figure 7.38 shows the one-way trip latencies for the same systems.

Network	Host Interfaces	Switch
Ethernet	≈$50	NA
ATM	≈$1000	≈$50,000 (ASX-200) ≈$30,000 (LattisCell 10114)

FIGURE 7.36 Approximate 1995 prices of measured networks.

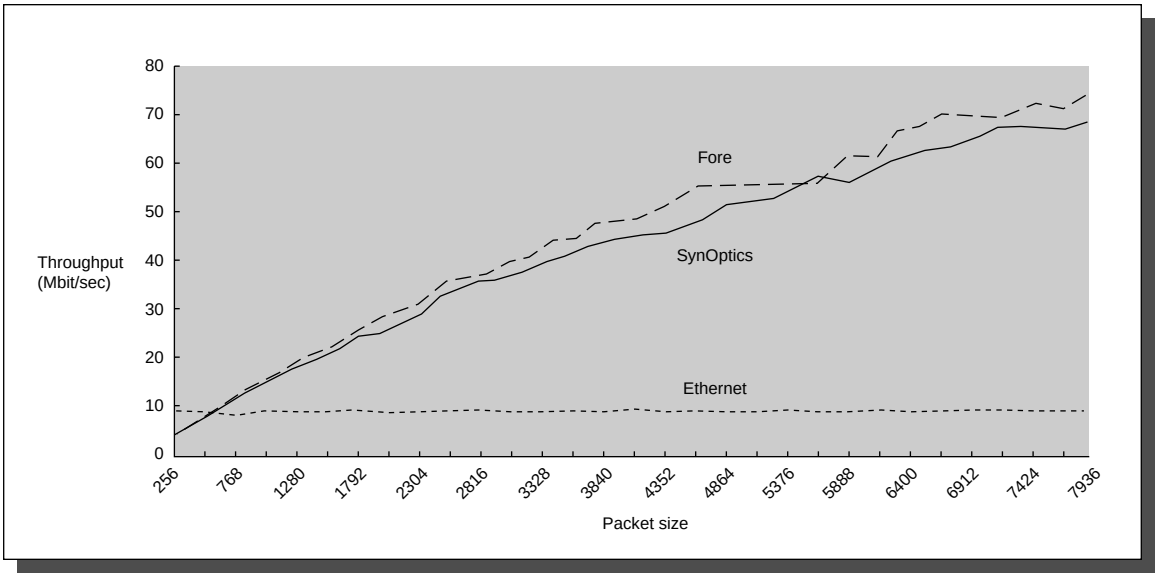

FIGURE 7.37 TCP throughput of ATM versus Ethernet between two machines as datagram size varies from 64 bytes to 8 KB. The Fore Systems switch offers roughly 5% to 10% higher bandwidth. Measurements were gathered by running user-level programs that exchanged data over TCP sockets. All measurements were done for point-to-point connections with little or no additional network traffic flowing through the switch. We measured the time to receive 5000 TCP packets and calculated the effective bandwidth (number of bytes received/elapsed time). Since these experiments measure user data, it does not give credit to the headers and trailers that are part of TCP, IP, AAL5, or ATM. If they were counted, the delivered throughput would be closer to 90 Mbits/second. (Measurements were taken by Kim Keeton of U.C. Berkeley.)

The most surprising result here is that the bandwidth for messages under 512 bytes is worse over the ATM network than it is over the Ethernet! The reason is that the vast majority of the time is spent in the operating system networking layers and the latency of the network itself is insignificant in comparison in both cases. The Ethernet turns out to be faster because its device drivers have been improved over many years and are thus better optimized.

To see the impact, let's estimate the times to replay the trace of NFS traffic from Figure 7.7 on page 573 on Ethernet and ATM. Measurements show the one-

Data size	Ethernet	ASX-200	LattisCell 10114
8	504	690	865
256	726	811	989
1024	1422	908	1083
4096	4174	1379	1589
8192	8631	1993	2274

FIGURE 7.38 One-way trip times in microseconds for different data payloads and networks. Measurements were gathered by running user-level programs that exchanged data over UDP sockets. All measurements were done for point-to-point connections with little or no additional network traffic flowing through the switch. The round-trip test was performed by sending messages over UDP in a request-response communication pattern. Reported above is one-half of the round-trip times. Single round-trip times were collected until a 95% confidence interval was obtained, or a minimum of 35 iterations were measured—whichever came first. (Measurements were taken by Kim Keeton of U.C. Berkeley.)

way time for the smallest packet on an unloaded Ethernet to be 504 microseconds, versus the 690 microseconds for ATM. We model the time as the fixed overhead per message plus a per-data-byte overhead. Our ATM measurements suggest a peak rate of 78 Mbits/sec using TCP/IP for the Fore ATM Switch and 9 Mbits/sec for Ethernet. Figure 7.39 shows the overhead time, transmission time, and total time to send all the NFS messages over Ethernet and ATM. The peak link speed of ATM is 15 times faster and the measured link speed for 8-KB datagrams is almost 9 times faster, but the higher overheads offset the benefits so that ATM would transmit these messages only 1.2 times faster.

Overall the performance is depressing: using a reliable transport protocol (TCP), we can utilize little more than half the network bandwidth. In addition, the small messages prevalent in request-response-style communication fare worse over ATM than over Ethernet. Why is this so? The answer is twofold: First, the UNIX networking layers are not designed for networks of this speed, and second, the driver-firmware interface used by Fore is inefficient.

The problem with the UNIX network layers is that they were designed a decade ago for much slower networks and workstations with much less memory. The same layers with the same algorithms are used for SLIP connections across 9.6 Kbits/second modems as well as for 155 Mbits/second ATM connections.

Another problem is caused by the fact that the driver-firmware interface is at the level of data blocks. This level requires the i960 to traverse descriptors and follow pointers in main memory using DMA accesses that are good for high bandwidth but not low latency. The result is that even if all the UNIX networking inefficiencies were alleviated, the round-trip times would still be several hundred microseconds. See page 623 for the fallacy concerning off-loading the main processor.

		Overhead (secs)			Transmission (secs)		Total time (secs)	
Size	No. messages	ATM	Ethernet	No. data bytes	ATM	Ethernet	ATM	Ethernet
32	771,060	532	389	33,817,052	4	48	536	436
64	56,923	39	29	4,101,088	0	5	40	34
96	4,082,014	2817	2057	428,346,316	46	475	2863	2532
128	5,574,092	3846	2809	779,600,736	83	822	3929	3631
160	328,439	227	166	54,860,484	6	56	232	222
192	16,313	11	8	3,316,416	0	3	12	12
224	4820	3	2	1,135,380	0	1	3	4
256	24,766	17	12	9,150,720	1	9	18	21
512	32,159	22	16	25,494,920	3	23	25	40
1024	69,834	48	35	70,578,564	8	72	56	108
1536	8842	6	4	15,762,180	2	14	8	19
2048	9170	6	5	20,621,760	2	19	8	23
2560	20,206	14	10	56,319,740	6	51	20	61
3072	13,549	9	7	43,184,992	4	39	14	46
3584	4200	3	2	16,152,228	2	14	5	17
4096	67,808	47	34	285,606,596	29	255	76	290
5120	6143	4	3	35,434,680	4	32	8	35
6144	5858	4	3	37,934,684	4	34	8	37
7168	4140	3	2	31,769,300	3	28	6	30
8192	287,577	198	145	2,390,688,480	245	2132	444	2277
Total	11,387,913	7858	5740	4,352,876,316	452	4132	8310	9872

FIGURE 7.39 Total time on Ethernet and ATM, calculating the total overhead and transmission time separately. Note that the size of the headers needs to be added to the data bytes to calculate transmission time. (NFS measurements taken by Mike Dahlin of U.C. Berkeley.)

7.11 Fallacies and Pitfalls

Interconnection networks are filled with myths and hazards; this section has just a few warnings, so proceed carefully.

Pitfall: Using bandwidth as the only measure of network performance.

Many network companies apparently believe that given sophisticated protocols like TCP/IP that maximize delivered bandwidth, there is only one figure of merit for networks. This may be true for some applications, such as video, where there

is little interaction between the sender and the receiver, but many applications, such as NFS, are of a request-response nature, and so for every large message there must be one or more small messages. Figure 7.39 shows that latency is as important as bandwidth.

Pitfall: Ignoring software overhead when determining message overhead.

Low software overhead requires cooperation with the operating system as well as with the communication libraries. As an example, the CM-5 has a software overhead of 20 μsecs to send a message and a hardware overhead of 0.5 microseconds. The Intel Paragon reduced the hardware overhead to just 0.2 microseconds, but the initial release of software has a software overhead of 250 microseconds. Later releases reduced this overhead to 25 microseconds, which still dominates the hardware overhead. Figure 7.39 shows a similar pitfall for ATM.

This pitfall is simply Amdahl's Law applied to networks: Faster network hardware is superfluous if there is not a corresponding decrease in software overhead.

Pitfall: Adding functions to interconnection systems that violate the end-to-end argument.

This argument is against providing features at a lower level that only partially satisfy the communication demand that can only be accomplished at the highest level. Saltzer, Reed, and Clark [1984] give the end-to-end argument as

> *The function in question can completely and correctly be specified only with the knowledge and help of the application standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. [page 278]*

Their example of the pitfall was a network at MIT that used several gateways, each of which added a checksum from one gateway to the next. The programmers of the application assumed the checksum guaranteed accuracy, incorrectly believing that the message was protected while stored in the memory of each gateway. One gateway developed a transient failure that swapped one pair of bytes per million bytes transferred. Over time the source code of one operating system was repeatedly passed through the gateway, thereby corrupting the code. The only solution was to correct the infected source files by comparing to paper listings and repairing the code by hand! Had the checksums been calculated and checked by the application running on the end systems, safety would have been assured.

Fallacy: Adding a processor to the network interface card improves performance.

Microprocessors make it very tempting to add intelligence to a network interface card. Like the pitfall in the last chapter (page 551), the danger is that the processor is much slower and that the additional processor will add to the latency of an operation. Here are two examples.

The Meiko CS-2 MPP uses a 66-MHz HyperSPARC microprocessor in the computation node and added a SPARC-compatible microprocessor built from gate arrays to the network interface card. Their custom design has a much slower clock rate and no caches, and hence the main processor is about 40 times faster. The Meiko's communication has higher overhead than the CM-5, which has simpler hardware despite the use of similar nodes on both MPPs.

To avoid the problem above, the Intel Paragon MPP uses the identical microprocessor twice: once for computation and once for communication. The two 50-MHz i860 XP microprocessors use a common bus that is cache-coherent, further simplifying the software model.

Liu and Culler [1995] measure the Paragon performance for sending eight words using two processors versus one processor. The latency for a single processor is 3.1 microseconds but grows to 9.1 microseconds for the dual processor! The basic reason for the slowdown is that extra work must be performed transferring the data between the cache of the communication microprocessor and the cache of the computation microprocessor. Sending a message requires the following steps on the Paragon:

1. The computation processor stores the messages into the message queue in shared memory for the communication processor to access.
2. The cache block for the queue is likely already in the communication processor, since there is little else for it to do. Since the i860 XP uses an invalidation-based protocol, the store by the first processor invalidates the block in the second. The communication processor is polling, so each time the computation processor writes one word, it is read into the cache of the communication processor and then invalidated by a subsequent write. There is also the extra cost of executing the instructions that write to memory and then read the message back. Each cache-to-cache transfer takes two bus transactions.
3. When the full message is in the cache of the communication processor, the processor sends it to the network interface card, which sends it over the network to the receiver's network interface card.
4. The receiver's communication processor loads the message into its registers.
5. Similar to step 2 above, there begins a sequence of stores by the communication processor and loads by the computation processor to move the data from one cache to the other, which may involve a repeated sequence of invalidates and partial copies.

The extra processor adds steps 2 and 5, which are the most time-consuming steps in the process.

The reasons designers add an extra processor are to avoid the overhead of invoking the operating system and to communicate in parallel with the computation. This works best for DMA-like transfers of large messages, but can interfere with latency of small messages.

Pitfall: Using TCP/IP as LAN protocol.

The network designers on the first workstations decided it would be elegant to use a single protocol stack no matter where the destination of the message: across a room or across an ocean, the TCP/IP overhead must be paid. This might have been a wise decision especially given the unreliability of early Ethernet hardware, but it sets a high software overhead barrier for commercial systems. Such an obstacle lowers the enthusiasm for low-latency network interface hardware and low-latency interconnection networks if the software is just going to waste hundreds of microseconds when the message must travel only dozens of meters.

TCP/IP advocates point out that the protocol itself is theoretically not as burdensome as the current implementations, but progress has been modest in systems shipped commercially.

7.12 Concluding Remarks

Networking is one of the most exciting fields in computer science and engineering today. The Internet and World Wide Web pervade our society and will likely revolutionize how we access information. Although we couldn't have the Internet without the telecommunication media, it is protocol suites such as TCP/IP that make electronic communication practical. More than almost any other area of computer science and engineering, these protocols embrace the concept that failures are the norm, and so the system must operate reliably in the presence of failures. Interconnection network hardware and software blend telecommunications with data communications, calling into question whether they should remain as separate academic disciplines or be combined into a single field.

The silicon revolution has made its way to the switch: just as the "killer micro" changed computing, whatever turns out to be the "killer network" will transform communication. We are seeing the same dramatic change in cost/performance in switches as the mainframe-minicomputer-microprocessor change did to processors. Inexpensive switches mean that network bandwidth can scale with the number of nodes, even in the local area network.

The combination of fast rate of improvement in processors and in switch-based LANs offers an interesting challenge to conventional computer design. Although the desktop computer is clearly the most cost-effective solution for problems that run fast enough on the desktop, there are many alternatives for problems that are bigger than the desktop. For example, both MPPs and shared-memory processors leverage the same microprocessor as on the desktop to provide larger-scale computing. The problem has been higher prices and delays of introduction for nodes in the multiprocessor versus the high volume and fast-moving desktop computer. The ability to upgrade nodes and switches independently may offer the same flexibility and competition in price and quality as are enjoyed by buyers of stereo components. In 1995, clusters or networks of workstations have the potential to pose a serious challenge to the other candidates for

large-scale computing. Challenges to be overcome include the high overhead of communication, the Achilles' heel of such collections of low-cost machines, and providing operating systems that can coordinate hundreds of machines [Anderson, Culler, and Patterson 1995].

The dramatic improvement in cost/performance of communications has enabled millions of people around the world to find others with common interests. We are not near any performance plateaus, so we expect rapid advance in both local and wide area networks. As the quotes at the beginning of this chapter suggest, the authors believe this revolution in two-way communication will change the form of human associations and actions.

7.13 Historical Perspective and References

This chapter has taken the unusual perspective that computers inside a cabinet of an MPP and computers on an intercontinental WAN share many of the same concerns. Although this observation may be true, their histories are very different.

Wide Area Networks

The earliest of the data interconnection networks are WANs. The forerunner of the Internet is the ARPANET, which in 1969 connected computer science departments across the U.S. that had research grants funded by the Advanced Research Project Agency (ARPA), a U.S. government agency. It was originally envisioned as using reliable communications at lower levels; it was the practical experience with failures of underlying technology that led to the failure-tolerant TCP/IP, which is the basis for the Internet today. Vint Cerf and Robert Kahn are credited with developing the TCP/IP protocols in the mid 1970s, winning the ACM Software Award in recognition of that achievement. Kahn [1972] is an early reference on the ideas of ARPANET.

In 1975 there were roughly 100 networks in the ARPANET and only 200 in 1983; in 1995 the Internet encompasses 50,000 networks worldwide, about half of which are in the United States. Interestingly, the key networks that made the Internet possible, such as ARPANET and NSFNET, have been replaced by fully commercial systems, and yet the Internet still thrives. The exciting application of the Internet is the World Wide Web, developed by a programmer at the European Center for Particle Research (CERN) in 1989 for information access. In 1992 a young programmer at the University of Illinois developed a graphical interface for Web called Mosaic, which became immensely popular. In May 1995 there are over 30,000 Web sites, and the number is doubling every two months.

One interesting sociological phenomenon is the ongoing battle between the advocates of ATM and the Internet protocols. The dream of ATM is a seamless software interface from the local area network to the wide area network, replacing the cumbersome TCP/IP protocol stack with a uniform protocol such as

AAL-5. The Internet advocates think ATM is a fine piece of underlying technology, but you still need TCP/IP because networks will continue to be heterogeneous: ISDN to the home, wireless to the portable, ATM to the office. Thus we are more likely to see contention than synergy between ATM and TCP/IP.

ATM is just the latest of the ongoing standards set by the telecommunications industry, and it is undoubtedly the future for this community. Communication forces standardization by competitive companies, sometimes leading to anomalies. For example, the telecommunication companies in North America wanted to use 64-byte packets to match their existing equipment, while the Europeans wanted 32-byte packets to match their existing equipment. The 48-byte compromise of the new standard was reached to ensure that neither group had an advantage in the marketplace!

Local Area Networks

ARPA's success with wide area networks led directly to the most popular local area networks. Many researchers at Xerox Palo Alto Research Center had been funded by ARPA while working at universities, and so they all knew the value of networking. This group invented the forerunner of today's workstations [Thacker et al. 1982] and Ethernets in 1974 [Metcalfe and Boggs 1976]. It was Boggs' experience as a ham radio operator that led to a design that did not need a central arbiter, but instead listened before use and then varied back-off times in case of conflicts.

This first Ethernet provided a 3 Mbits/sec interconnection, which seemed like an unlimited amount of communication bandwidth with computers of that era, relying on the interconnect technology developed for the cable television industry. The announcement by Digital Equipment Corporation, Intel, and Xerox of a standard for 10 Mbits/sec Ethernet was critical to the commercial success of Ethernet. This announcement short-circuited a lengthy IEEE standards effort, which eventually did publish IEEE 802.3 as an standard for Ethernets.

It is long past time to replace the Ethernet, and there have been several unsuccessful candidates. Unfortunately, the FDDI committee took a very long time to agree on the standard and the resulting interfaces were expensive. It is also a shared medium when switches are becoming affordable.

As mentioned earlier, the failure of FDDI on the desktop has led the LAN community to look elsewhere, and at the time of this writing there are three promising candidates: switched Ethernet, 100-Mbit Ethernet, and ATM.

Massively Parallel Processors

The final component of interconnect networks is found in massively parallel processors (MPPs). One forerunner of today's MPPs is the Cosmic Cube [Seitz 1985], which used Ethernet interface chips to connect 8086 computers in a hypercube. MPP interconnections have improved considerably since then, with

messages routed automatically through intermediate switches to their final destinations at high bandwidths and with low latency. Considerable research has gone into the benefits over different topologies in both construction and program behavior. Whether due to faddishness or changes in technology is hard to say, but topologies certainly become very popular and then disappear. The hypercube, widely popular in the 1980s, has almost disappeared from MPPs of the 1990s. Cut-through routing, however, has been preserved and is covered by Dally and Seitz [1986].

By the second edition of this book, MPPs have fallen on hard times: Thinking Machines, Kendall Square Research, and Cray Computer Corporation declared bankruptcy, and Intel Supercomputer announced a reorganization. Although the Cray T3D and N-cube MPP continue unchanged, the MPP standard bearer in 1995 is clearly the IBM SP-2. It uses processor boards from the RS/6000 model 590 workstation, the AIX operating systems from the workstations, and a switch from an MPP research project. The conventionally configured maximum size of the SP-2 is 128 nodes, with two special orders at 512 nodes. The M of MPP stands for massive, and massive is certainly smaller in 1995 than it was in the 1980s.

The Future

At the time of writing this second edition, a new class of networks is emerging: *system area networks*. These recent networks are designed for a single room or single floor and thus the length is ten to hundreds of meters, falling between an MPP interconnection network and a LAN. Close distance means the wires can be wider and faster at lower cost, network hardware can ensure error-free delivery, and less handshaking time is consumed when cascading switches together. There is also less reason to go to the cost of optical fiber, since the distance advantage of fiber is less important for SANs. The limited size of the networks also makes source-based routing plausible, further simplifying the network. Both Tandem Computers and Myricom sell SANs.

Will ATM, Ethernet successors, or SANs be the killer network of the next decade? In 1995 it is very hard to tell. A wonderful characteristic of computer architecture is that such issues will not remain academic debates, unresolved as people rehash the same arguments time and again. Instead, the battle will be fought in the marketplace, with well-funded and talented groups giving their best shots at shaping the future. The best combination of technology and follow-through has often determined commercial success.

Thus time will tell us who wins and who loses; we shall know the score by the next edition of this text.

References

ALLES, A. [1993]. *ATM in Private Networking*, Interop 93 Tutorial.

ANDERSON, T. E., D. E. CULLER, D. PATTERSON [1995]. "A case for NOW (networks of workstations)," *IEEE Micro* 15:1 (February), 54–64.

ARPACI, R. H., D. E. CULLER, A. KRISHNAMURTHY, S. G. STEINBERG, AND K. YELICK [1995]. "Empirical evaluation of the CRAY-T3D: A compiler perspective," *Proc. 23rd Int'l Symposium on Computer Architecture* (June), Italy.

ATM FORUM [1994]. *ATM User-Network Interface Specification: Version 3.1*., PTR Prentice Hall, Englewood Cliffs, N.J.

BREWER, E. A. AND B. C. KUSZMAUL [1994]. "How to get good performance from the CM-5 data network." *Proc. Eighth Int'l Parallel Processing Symposium* (April), Cancun, Mexico.

COMER, D. [1993]. *Internetworking with TCP/IP*, 2nd ed., Prentice Hall, Englewood Cliffs, N.J.

DALLY, W. J. AND C. I. SEITZ [1986]. "The torus routing chip," *Distributed Computing* 1:4, 187–96.

DESURVIRE, E. [1992]. "Lightwave communications: The fifth generation," *Scientific American* (International Edition) 266:1 (January), 96–103.

KAHN, R. E. [1972]. "Resource-sharing computer communication networks," *Proc. IEEE* 60:11 (November), 1397–1407.

LIU, L. T. AND D. E. CULLER [1994]. "Measurement of active message performance on the CM-5," Tech. Rep. UCB/CSD94-807, University of California, Berkeley.

LIU, L. T. AND D. E. CULLER [1995]. "An evaluation of the Intel Paragon communication architecture," submitted to *Supercomputing 95*, San Diego, Calif.

METCALFE, R. M. [1993]. "Computer/network interface design: Lessons from Arpanet and Ethernet. *IEEE J. on Selected Areas in Communications* 11:2 (February), 173–80.

METCALFE, R. M. AND D. R. BOGGS [1976]. "Ethernet: Distributed packet switching for local computer networks," *Comm. ACM* 19:7 (July), 395–404.

PARTRIDGE, C. [1994]. *Gigabit Networking*. Addison-Wesley, Reading, Mass.

SALTZER, J. H., D. P. REED, D. D. CLARK [1984]. "End-to-end arguments in system design," *ACM Trans. on Computer Systems* 2:4 (November), 277–88.

SEITZ, C. L. [1985]. "The Cosmic Cube (concurrent computing),"*Communications of the ACM* 28:1 (January), 22–33.

TANENBAUM, A. S. [1988]. *Computer Networks*, 2nd ed., Prentice Hall, Englewood Cliffs, N.J.

THACKER, C. P., E. M. MCCREIGHT, B. W. LAMPSON, R. F. SPROULL, AND D. R. BOGGS [1982]. "Alto: A personal computer," in *Computer Structures: Principles and Examples*, D. P. Siewiorek, C. G. Bell, and A. Newell, eds., McGraw-Hill, New York, 549–572.

WALRAND, J. [1991]. *Communication Networks: A First Course*, Aksen Associates: Irwin, Homewood, Ill.

EXERCISES

7.1 [15] <7.2> Assume the overhead to send a zero-length data packet on an Ethernet is 500 microseconds and that an unloaded network can transmit at 90% of the peak 10 Mbits/sec rating. Plot the delivered bandwidth as the data transfer size varies from 32 bytes to 1500.

7.2 [15] <7.2> One reason that ATM has a fixed transfer size is that when a short message is behind a long message, a node may need to wait for an entire transfer to complete. For applications that are time-sensitive, such as when transmitting voice or video, the large transfer size may result in transmission delays that are too long for the application. On an unloaded interconnection, what is the worst-case delay if a node must wait for one full-size Ethernet packet versus an ATM transfer? See Figure 7.22 (page 603) to find the packet sizes. For this question assume you can transmit at 100% of the 155 Mbits/sec of the ATM network and 100% of the 10 Mbits/sec Ethernet.

7.3 [20/10] <7.4>Is electronic communication always fastest for longer distances than the Example on page 579? Calculate the time to send 100 GB using 10 8-mm tapes and an overnight delivery service versus sending 100 GB by FTP over the Internet. Make the following four assumptions:

- The tapes are picked up at 4 P.M. Pacific time and delivered 4200 km away at 10 A.M. Eastern time (7 A.M. Pacific time).
- On one route the slowest link is a T1 line, which transfers at 1.5 Mbits/sec.
- On another route the slowest link is a 10 Mbits/sec Ethernet.
- You can use 50% of the slowest link between the two sites.

a. [20] <7.4> Will all the bytes sent by either Internet route arrive before the overnight delivery person arrives?

b. [10] <7.4> What is the bandwidth of overnight delivery? Calculate the average bandwidth of overnight delivery service for a 100-GB package.

7.4 [20/20/20/20] <7.9> If you have access to a UNIX system, use `ping` to explore the Internet. First read the manual page. Then use `ping` without option flags to be sure you can reach the following sites. It should say that `X is alive`. Depending on your system, you may be able to see the path by setting the flags to verbose mode (`-v`) and trace route mode (`-R`) to see the path between your machine and the example machine. Alternatively, you may need to use the program `traceroute` to see the path. If so, try its manual page. You may want to use the UNIX command `script` to make a record of your session.

a. [20] <7.9> Trace the route to another machine on the same local area network.

b. [20] <7.9> Trace the route to another machine on your campus that is *not* on the same local area network.

c. [20] <7.9> Trace the route to another machine *off campus*. For example, if you have a friend you send email to, try tracing that route. See if you can discover what types of networks are used along that route.

d. [20] <7.9> One of the more interesting sites is the McMurdo NASA government station in Antarctica. Trace the route to mcmvax.mcmurdo.gov.

7.5 [12/15/15] <7.5> Assume 64 nodes and 16×16 ATM switches in the following. (This exercise was suggested by Mark Hill.)

a. [12] <7.5> Design a switch topology that has the minimum number of switches.

b. [15] <7.5> Design a switch topology that has the minium latency through the switches. Assume unit delay in the switches and zero delay for wires.

c. [15] <7.5> Design a switch topology that balances the bandwidth required for all links. Assume a uniform traffic pattern.

7.6 [20] <7.5> Redo the cut-through routing calculation for CM-5 on page 594 of different sizes: 64, 256, and 1024 nodes.

7.7 [15] <7.5> Calculate the time to perform a broadcast (from-one-to-all) on each of the topologies in Figure 7.17 on page 587, making the same assumptions as the two Examples on pages 588–590.

7.8 [20] <7.5> The two Examples on pages 588–590 assumed unlimited bandwidth between the node and the network interface. Redo the calculations in Figure 7.17 on page 587, this time assuming a node can only issue one message in a time unit.

7.9 [15] <7.5> Compare the interconnection latency of a crossbar, Omega network, and fat tree with eight nodes. Use Figure 7.13 on page 584 and add a fat tree similar to Figure 7.14 on page 585 as a third option. Assume that each switch costs a unit time delay. Assume the fat tree randomly picks a path, so give the best case and worst case for each example. How long will it take to send a message from node P0 to P6? How long will it take P1 and P7 to also communicate?

7.10 [15] <7.5> One interesting measure of the latency and bandwidth of an interconnection is to calculate the size of a message needed to achieve one-half of the peak bandwidth. This halfway point is sometimes referred to as $n_{1/2}$, taken from the vector processing. Using Figure 7.37 on page 620, estimate $n_{1/2}$ for TCP/IP message using ATM and the Ethernet.

7.11 [15] <7.9> Use FTP to transfer a file from a remote site and then between local sites on the same LAN. What is the difference in bandwidth for each transfer? Try the transfer at different times of day or days of the week. Is the WAN or LAN the bottleneck?

7.12 [15] <7.5> Draw the topology of a 6-cube similar to the drawing of the 4-cube in Figure 7.16 on page 587.

7.13 [12/12/12/15/15/18] <7.7> Use M/M/1 queuing model to answer this exercise. Measurements of a network bridge show that packets arrive at 200 packets per second and that the gateway forwards them in about 2 ms.

a. [12] <7.7> What is the utilization of the gateway?

b. [12] <7.7> What is the mean number of packets in the gateway?

c. [12] <7.7> What is the mean time spent in the gateway?

d. [15] <7.7> Plot the response time versus utilization as you vary the arrival rate.

e. [15] <7.7> For an M/M/1 queue, the probability of finding n or more tasks in the system is Utilization^n. What is the chance of an overflow of the FIFO if it can hold 10 messages?

f. [18] <7.7> How big must the gateway be to have packet loss due to FIFO overflow to be less than one packet per million?

7.14 [20] <7.7> The imbalance between the time of sending and receiving can cause problems in network performance. Sending too fast can cause the network to back up and increase the latency of messages, since the receivers will not be able to pull out the message fast enough. A technique called *bandwidth matching* proposes a simple solution: Slow down the sender so that it matches the performance of the receiver [Brewer 1994]. If two machines exchange an equal number of messages using a protocol like UDP, one will get ahead of the other, causing it to send all its messages first. After the receiver puts all these messages away, it will then send its messages. Estimate the performance for this case versus a bandwidth-matched case. Assume the send overhead is 200 microseconds, the receive overhead is 300 microseconds, time of flight is 5 microseconds, and latency is 10 microseconds, and that the two machines want to exchange 100 messages.

7.15 [40] <7.7> Compare the performance of UDP with and without bandwidth matching by slowing down the UDP send code to match the receive code as advised by bandwidth matching [Brewer 1994]. Devise an experiment to see how much performance changes as a result. How should you change the send rate when two nodes send to the same destination? What if one sender sends to two destinations?