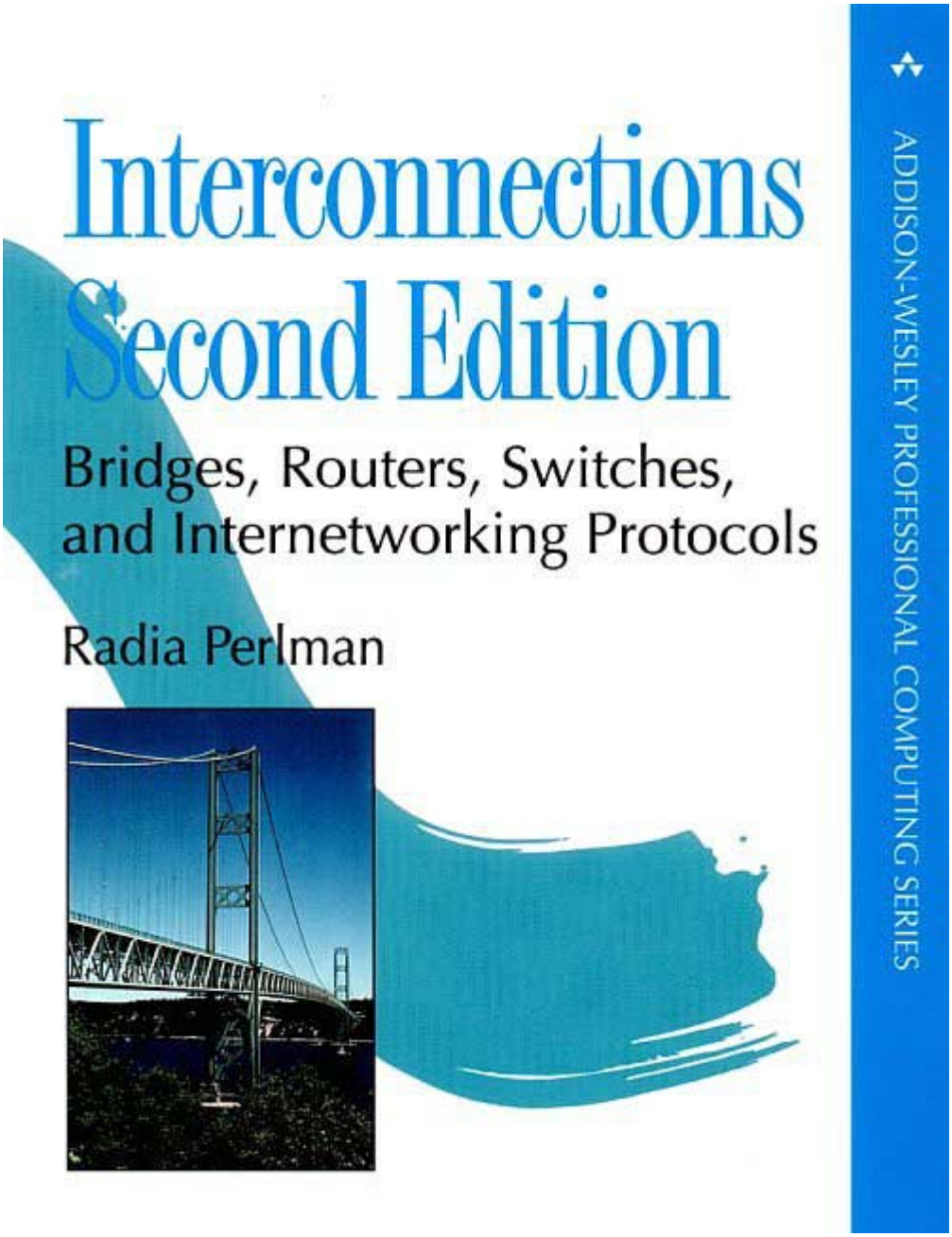


Preface

Interconnections, Second Edition is about what goes on inside the boxes that move data around the Internet. These boxes are variously called bridges, routers, switches, and hubs. The book also describes the devices that connect to the network.

There is considerable confusion in this area. Most of the terminology is ill defined and is used in conflicting ways. The terminology and the specifications tend to be daunting. Some knowledge is spread among many different documents; much is unwritten folk wisdom. Adding to the confusion is dogma. Beliefs are accepted as truth, and questioning any of the dogma is often greeted with hostility. But good engineering demands that we understand what we're doing and why, keep an open mind, and learn from experience.

In *Interconnections, Second Edition*, instead of diving right into the details of one protocol, I first focus on the problems to be solved. I examine various solutions to each of these problems and discuss the engineering trade-offs involved. Then I look at a variety of solutions that have been deployed and compare the approaches. I give technical arguments for any opinions, and if you think I have missed any arguments I welcome email discussion. My email

address is at the back of the book, which I hope you will find after having read the book cover to cover.
In the first edition, my intention was to help people understand the problems and the general types of solutions, assuming that they would read the specifications to get the details of specific protocols. But people used the book as a reference in addition to using it to understand the issues. So in this edition I have documented many more of the protocols in detail.
I believe that to understand something deeply you need to compare it to something else. The first edition was "minimalist" in that I always used only two examples: two types of bridges, bridges versus routers, connection-oriented versus connectionless network layer protocols, and two examples of connectionless protocols (CLNP and IP). In this edition I add a lot more examples, including ATM, IPv6, IPX, AppleTalk, and DECnet. I did this in part because these protocols exist, and it is hard to get information about them. But mostly I did it because the protocols embody interesting ideas that should not be lost. When we design new protocols, we should learn from previous ideas, both good and bad. Also, it takes very little additional effort, after the problem is described generically, to describe several examples.
The Tao of network protocols: If all you see is IP, you see nothing.
—Greg Minshall

Roadmap to the Book

The first four chapters are not significantly different from their counterparts in the first edition, but the rest of the book has been largely rewritten. Chapters 1 through 4 cover general networking concepts, data link issues such as addressing and multiplexing, transparent bridges and the spanning tree algorithm, and source routing bridges. Chapter 5 is completely new and explains how the notion of a switch evolved into a rediscovery of the bridge. It also covers VLANs and fast Ethernet.
The remainder of the book concentrates on layer 3 (the network layer). Chapter 6 gives an overview of the network layer. Chapter 7 covers connection-oriented networks, including ATM and X.25. Chapter 8 discusses the issues in a generic connectionless network layer. Chapter 9 covers layer 3 addressing generically and gives a detailed comparison of IP, IPv6, CLNP, DECnet, AppleTalk, and IPX. Chapter 10 covers the information that should appear in a network layer header and contrasts the headers of several protocols.
Chapter 11 covers autoconfiguration and neighbor discovery, including protocols such as ARP and DHCP. Chapter 12 covers routing algorithms generically.
Chapter 13 discusses the problem of doing longest-prefix matching, which is required in order to forward IP packets quickly. Chapter 14 discusses the specifics of various routing protocols including RIP, IS-IS, OSPF, PNNI, NLSP, and BGP. Chapter 15 covers network layer multicast. Chapter 16 explains how to design a network that is invulnerable to sabotage, an idea whose time may come.
The final two chapters summarize the book, and I hope they will be mostly light and entertaining reading. Chapter 17 probes the mystery of what, if anything, distinguishes a router from a bridge. Chapter 18 attempts to capture folk wisdom about how to design a protocol.
Finally, there is an extensive glossary. I try to define terms when I first use them, but if I ever fail to do that, you will probably find them in the glossary.

Acknowledgments

Writing this section is scary because I am afraid I will leave people out. I'd like to thank the people who reviewed all or part of the book: Peter Memishian, Paul Koning, Tony Lauck, Craig Partridge, Dan Pitt, Brian Kernighan, Paul Bottorff, Joel Halpern, Charlie Kaufman, Mike Speciner, Andy Tanenbaum, Phil Rosenzweig, Dan Senie, William Welch, Craig Labovitz, Chase Bailey, George Varghese, and Suchi Raman. Other people who have been helpful by answering questions are Ariel Hendel, Rich Kubota, Stuart Cheshire, Tom Maufer, Steve Deering, and John Moy. The first time I sent an email question in the middle of the night (when I did most of my work on this book) to Craig Partridge, the co-series editor for this

book, the beep indicating incoming mail happened so immediately that I assumed it was an automatic mail responder informing me he was on vacation. But it was an answer to my question. I assume he doesn't have an automatic mail responder so clever that it can answer technical questions, so I thank him for being so prompt and available. Brian Kernighan, the other series editor, also had detailed and helpful comments on the entire book.
The people at Addison-Wesley have been amazingly patient with me for the many years in which I've been working on this edition. I'm not sure they had any alternative besides patience, but it was nice that they believed I'd finish even when I wasn't so sure. So thank you to Mary Hart, Karen Gettman, Jacquelyn Doucette, and Jason Jones. And I'd also like to thank my copy editor, Betsy Hardinger. She of all people will have read every word of the book, while maintaining the concentration to note inconsistencies and ways of removing excess words here and there. I know it's her job, but I'm still impressed.
Mike Speciner helped me figure out the mysteries of Framemaker. Ray Perlner made sure that I maintained some humor in the book and watched over my shoulder while I typed the last chapter to see that I had enough funny bad real-life protocols. Dawn Perlner has been terrifically supportive, convincing her friends and even strangers in bookstores to buy my books. She used to be my child. Now she's a wonderful friend.

About the Author

Radia Perlman's work has had a profound impact on the field of networking. She was featured in the 25th anniversary edition of *Data Communications* magazine as one of the 25 people whose work has most influenced the industry. She is the creator of the spanning tree algorithm upon which bridges (switches) are based, and the algorithms that allow robust and efficient link state routing, upon which all modern link state routing protocols (such as IS-IS, OSPF, and PNNI) are based. Radia designed IS-IS, Simple Multicast, and sabotage-proof routing. She is also co-author of *Network Security: Private Communication in a Public World*. Both of her books were listed in the top 10 most useful networking reference books in the March, 1998 issue of *Network Magazine*. She is currently a Distinguished Engineer at Sun Microsystems, Inc. She holds 50 patents and a Ph.D. from M.I.T.

Chapter 1. Essential Networking Concepts

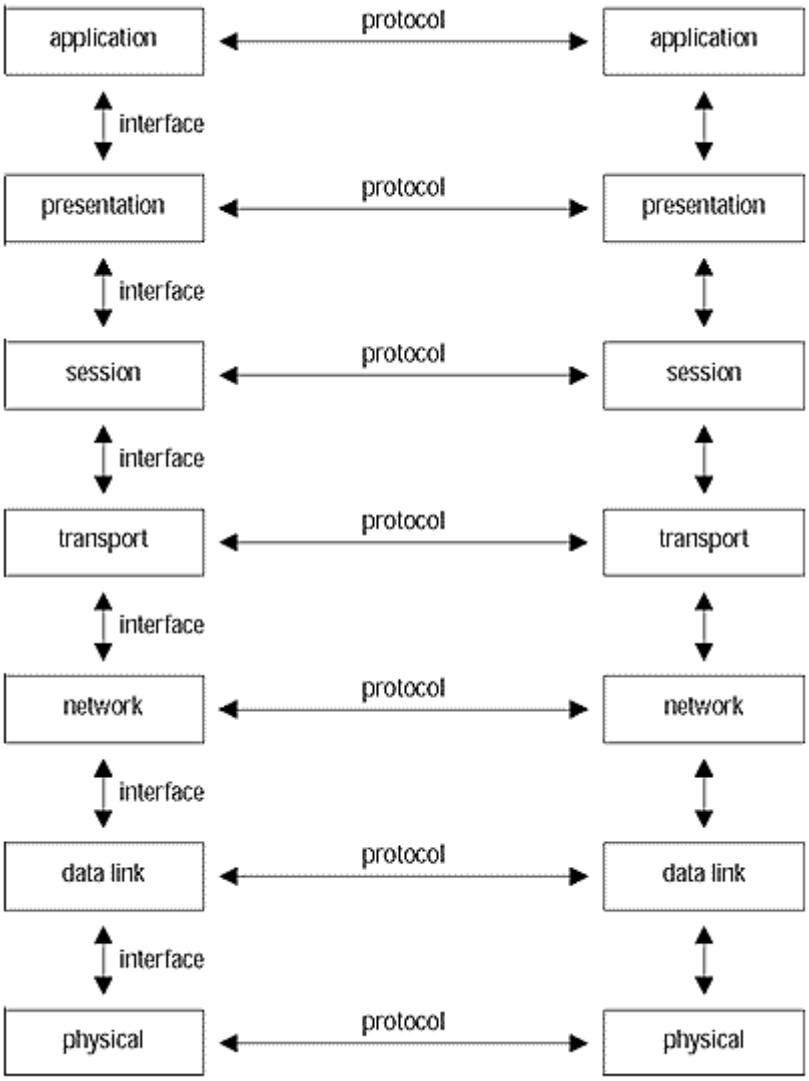
This chapter introduces concepts that are essential to understanding the specific subfield of computer networking that includes bridges and routers. It covers the OSI reference model, including layering and service models, because this model is a useful basis for some vocabulary. It also discusses various dimensions along which network designs can differ, such as scope, scalability, robustness, and autoconfigurability. Chapter 1 also describes the typical techniques involved in providing reliable two-party communication because some of the techniques used by routers can interact with techniques used by other layers.

1.1 Layers

Understanding, designing, and building a computer network would be too difficult a task unless the problem were partitioned into smaller subtasks, traditionally by dividing the problem into layers. The idea behind layering is that each layer is responsible for providing a service to the layer above by using the services of the layer below.
Each layer communicates with its *peer* layer in another node through the use of a *protocol.* This communication is accomplished through direct communication with the layer below. The communication between layer *n* and layer *n*–1 is known as an *interface.*
The OSI (Open Systems Interconnection) Reference Model defines seven layers, as shown in Figure 1.1. There is nothing magic about the number seven or the functionality in the layers. The reference model was designed before the protocols themselves, and then committees were set up to design each of the layers. Many of the layers were subsequently subdivided into further layers. The distinction between the layers is not always clear. Bridges and routers are a good example of a case in which people should rightfully be confused about which

layers are which. But semantic arguments about layers are not very productive. Instead, the layering should be viewed as a useful framework for discussion and not as a bible.

Figure 1.1. The OSI Reference Model



Layers defined by ISO:

1. **Physical layer:**

   The physical layer transmits bits of information across a link. It deals with such problems as size and shape of connectors, assignment of functions to pins, conversion of bits to electrical signals, and bit-level synchronization. It is usual for several different types of physical layers to exist within a network and even for multiple different types of physical layers to exist within a node, because each technology requires its own physical layer.

2. **Data link layer:**

   The data link layer (sometimes called the link layer) transmits chunks of information across a link. It deals with such problems as checksumming to detect data corruption; coordinating the use of shared media, as in a LAN (local area network); and addressing (when multiple systems are reachable, as in a LAN). Again, it is common

for different links to implement different data link layers and for a node to support several data link layer protocols, one for each of the types of links to which the node is attached.

3. **Network layer:**

   The network layer enables any pair of systems in the network to communicate with each other. A *fully connected* network is one in which every pair of nodes has a direct link between its nodes, but this kind of topology does not scale beyond a few nodes. In the more typical case, the network layer must find a path through a series of connected nodes, and nodes along the path must forward packets in the appropriate direction. The network layer deals with such problems as route calculation, packet fragmentation and reassembly (when different links in the network have different maximum packet sizes), and congestion control.

4. **Transport layer:**

   The transport layer establishes a reliable communication stream between a pair of systems. It deals with errors that can be introduced by the network layer, such as lost packets, duplicated packets, packet reordering, and fragmentation and reassembly (so that the user of the transport layer can deal with larger-size messages and so that less-efficient network layer fragmentation and reassembly might be avoided). It is also nice if the transport layer reacts to congestion in the network by sending data more slowly in response.

5. **Session layer:**

   ISO had something in mind for this layer that doesn't seem useful to the Internet community. ISO's session layer offers services beyond the simple full-duplex reliable communication stream provided by transport, such as dialogue control (enforcing a particular pattern of communication between systems) and chaining (combining groups of packets so that either all or none of the packets in the group gets delivered). Whatever this layer is, it's irrelevant for bridges and routers.

6. **Presentation layer:**

   The goal of this layer is to agree on representations for data so that people defining structures don't have to worry about bit/byte order or what a floating point number looks like. ISO standardized on ASN.1 (Abstract Syntax Notation 1). Although I have yet to meet anyone who actually *likes* ASN.1—because it is complex and inefficient in both space and processing—a lot of the IETF (Internet Engineering Task Force) standards use it.

7. **Application layer:**

   As fascinating as bridging and routing is, it's actually because of applications that people want any of this stuff. Applications include file transfer, virtual terminal, Web browsing, and so on. It is common for multiple applications to be running concurrently in a node.

In this book, the data link layer is relevant because bridges operate within it and because the service provided by it is relevant to routers, which operate at the network layer, thereby making the network layer also relevant. The transport layer is somewhat relevant because it is a user of the network layer and because certain decisions that the network layer might make (such as whether to allow traffic to be split among several equivalent paths) affect the transport layer. The layers above transport are largely irrelevant to the study of bridges and routers.

Typically, the way layer *n* works is that it receives a chunk of data from layer *n*+1 along with additional information (such as the destination address) that might be required. Layer *n* must transmit the data to the layer *n* process in the destination node, which delivers it to the layer *n*+1 process in the destination node. Layer *n* often needs to include with the data certain information—for example, the address of the destination—that will be interpreted by other layer *n* entities. To get the information to the destination node, layer *n* hands down a buffer to layer *n*–1, including the data received from layer *n*+1 and the control information added by layer *n.* Additionally, layer *n* might pass other information in the layer *n n*–1 interface along with the buffer.

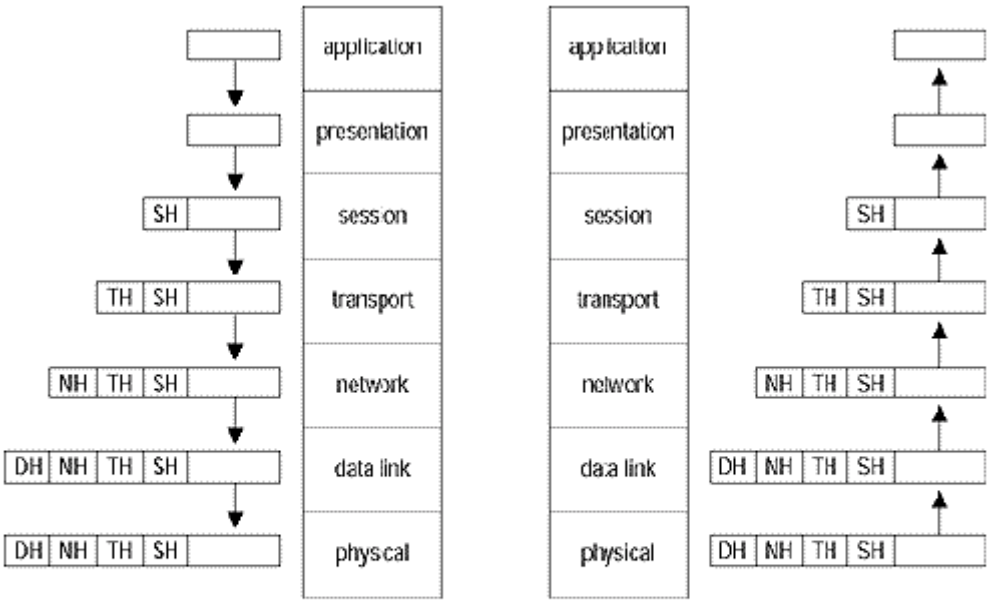
Let's look at an example of how layering works. Assume that the physical layer allows a stream of bits to pass from one machine to another. The data link layer marks the bit stream so that the beginning and end of a packet can be found; it also adds a checksum to the packet so that the receiving machine can detect whether noise on the line introduced errors. There are various interesting techniques to ensure that the marker indicating the beginning or end of the packet does not appear inside the data. In one technique, known as *bit stuffing,* the marker is a sequence of six 1's. To ensure that six consecutive 1's do not appear in the data portion of a packet, the transmitter adds an extra 0 after five consecutive 1's. The receiver knows that if the next bit after five consecutive 1's is a 0, then the 0 should be removed and ignored. If the next bit after five consecutive 1's is a 1, then it is a signal for the beginning or end of a packet. Another technique involves using different physical signals for data bits (1's and 0's) than for markers.

The network layer allows communication across multiple hops by cooperating with the network layers in all the connected machines to compute routes.

When the network layer receives a packet from the transport layer for transmission, the network layer adds an *envelope,* which consists of information glued onto the beginning (known as a *header*) and/or onto the end (known as a *trailer*). The envelope includes information such as the source and destination addresses. The network layer chooses an appropriate link on which to dispatch the packet and then hands the packet plus the network layer envelope to the data link layer process responsible for the outgoing link.

When the packet is received by an intermediate node, it is processed by the data link layer. Then the data link layer envelope is removed and the packet is passed up to the network layer, where the packet looks exactly the way it did when the previous network layer handed the packet to the data link layer—that is, it has everything transport sent down plus the network layer envelope. The network layer process at the receiving node looks at the network layer envelope, makes a decision as to the direction in which the packet should go based on that envelope, modifies the envelope as necessary (for example, incrementing a hop count to indicate how many nodes the packet has passed through), and gives the modified packet to the data link layer process responsible for the outgoing link (see Figure 1.2).

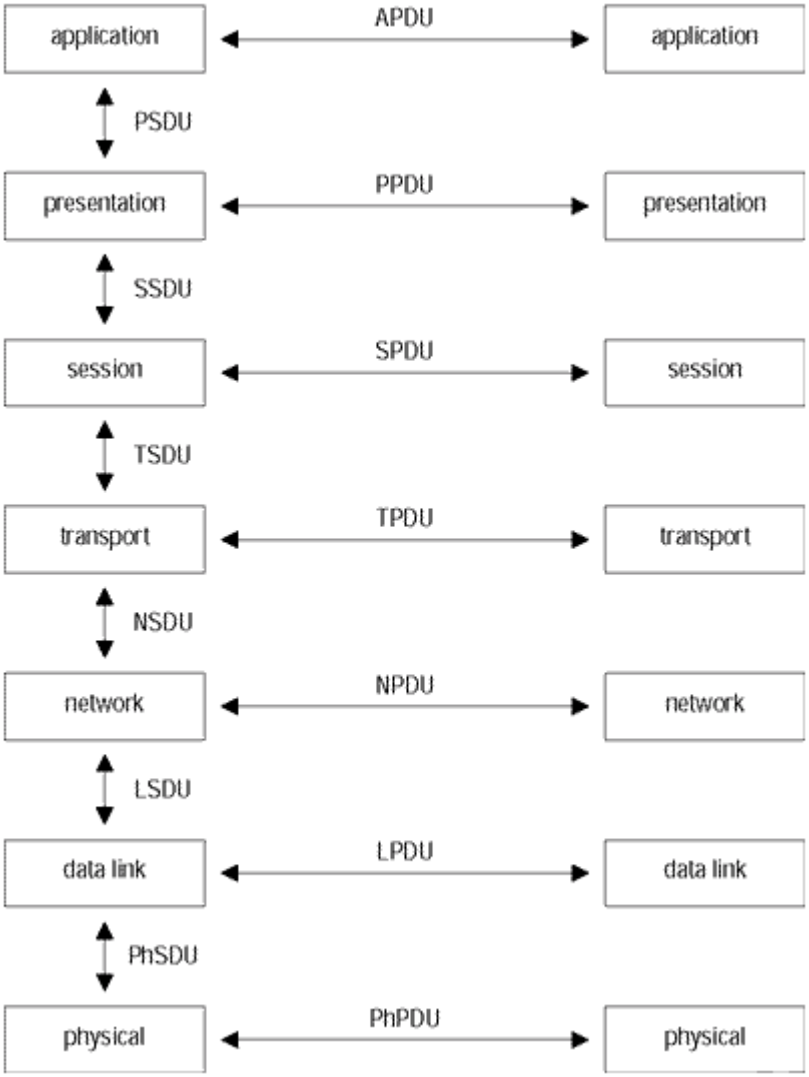
Figure 1.2. Envelopes added by lower layers



In the preceding description, words such as *packet* can be confusing. ISO has invented terminology that makes everything precise. Each layer communicates with its peer layer through a *protocol data unit,* or PDU. To make it clear which layer is being discussed, a single-letter prefix is added to PDU. The data link layer communicates with a peer data link layer by transmitting LPDUs. The network layer communicates with other network layers through NPDUs. The transport layer communicates with other transport layers through TPDUs.

When layer *n*+1 gives information to layer *n* for transmission, the information is known as an SDU, or *service data unit.* As with PDUs, a single-letter prefix is added to eliminate ambiguity. When the transport layer wishes to transmit a TPDU to another transport layer, it must do so by giving the network layer an NSDU. The network layer takes the NSDU, adds an envelope, and transmits it (through the data link layer) as an NPDU (see Figure 1.3).

Figure 1.3. PDUs and SDUs



As a rule, the ISO terminology is not used in this book because it is wordy and hard to translate "in real time" until one has attended at least three standards meetings. However, the ISO terminology is used occasionally when it is necessary to be very precise.

1.2 Service Models

In general, the service provided by layer *n*–1 for layer *n* consists of transmitting data. Layer *n* provides layer *n*–1 with data (an SDU) plus extra information, such as the address of the destination. Layer *n* must also be able to receive data from a peer layer *n,* which it does by having layer *n*–1 inform it that data is available.
Layer *n*–1 can provide either connectionless or connection-oriented service. A *connectionless* service offers layer *n* two functions: it accepts packets for transmission from layer *n,* or it delivers packets to layer *n.*
With a *connection-oriented* service, a connection must be established before data can be transferred. Communication consists of three phases:

1. Connection setup
2. Data transfer (transmission or receipt of data)
3. Connection release

Associated with each of these phases are two functions: one in which layer *n* initiates the function and another in which layer *n*–1 informs layer *n* that its peer has initiated the function.

For connection setup, either layer *n* requests that a connection be set up to some destination address, or layer *n*–1 informs layer *n* that some layer *n* process in some other node is requesting a connection.
For connection release, either layer *n* requests that a connection be released, or layer *n*–1 informs layer *n* that the layer *n* process in the other node (or some other condition) requires release of the connection.
Various interfaces might provide other functions, but the preceding are the basic ones.
Services can also vary in their degree of reliability. A service that is purely *datagram* (also known as *best-effort*) accepts data but makes no guarantees as to delivery. Data may be lost, duplicated, delivered out of order, or mangled. A *reliable* service guarantees (or claims to guarantee) that data will be delivered in the order transmitted, without corruption (mangling of the bits), duplication, or loss. It is possible to build a connection-oriented network that is reliable (for example, X.25) or datagram (for example, ATM, Asynchronous Transfer Mode). A connectionless network layer only makes sense offering a datagram service because by definition nothing is keeping track of what gets delivered. IP, IPX, DECnet, CLNP, and AppleTalk are examples of connectionless network layers. Figure 1.4 shows examples of types of network layers.

Figure 1.4. Examples of types of network layers

	connection-oriented	connectionless
datagram	ATM	IP, IPX, DECnet
reliable	X.25	

Intuitively, it might seem strange to want anything but a reliable service. However, reliability is not free. It usually makes layer *n*–1 more costly and less efficient. Although there are sometimes good reasons for providing reliability at more than one layer, it certainly need not be provided at every layer.
The trade-offs between connection-oriented and connectionless service and between datagram and reliable service are discussed in Chapter 6. In the world of the ISO and IEEE (Institute of Electrical and Electronics Engineers) standards, the advocates of connection-oriented, reliable service and the advocates of connectionless, datagram service can never convince each other, so both types of service tend to be offered at various layers in ISO and IEEE protocols. As you will see in the discussion of LANs in Chapter 2, this is why two flavors of data link layer are offered for running over LANs. The service is known as LLC, or *logical link control.* LLC type 1 is a connectionless datagram service; LLC type 2 is a reliable, connection-oriented service. ISO wound up defining two network layers: CONS (connection-oriented network service) and CLNS (connectionless network service).
Because there is no agreement about the kind of service provided by the network layer, five classes of transport are defined by ISO. They range from class 0, known as TP0, in which the assumption is made that the network layer does almost everything, to class 4, known as TP4, in which the assumption is made that the network layer is a datagram service.
In the TCP/IP protocol suite, the network layer (Internet Protocol, or IP) is connectionless. There are two transport layers. TCP (Transmission Control Protocol) offers reliable connection-oriented service, and UDP (User Datagram Protocol) offers datagram service.
ATM offers a connection-oriented, unreliable service that can be viewed as a network layer. But if there is another network layer—for example IP—running over ATM, then ATM is viewed by IP as a data link layer. As I said before, don't take the layering too seriously.
There is more discussion of service models in Chapter 6, including issues such as performance guarantees.

1.3 Important Properties of a Network

It is possible for network designs to seem as if they offer equivalent functionality. However, there are subtle ways in which they might differ. When you evaluate a network architecture, you should consider the following properties.

1. **Scope:**

   A network architecture should solve as general a problem as possible. It should be designed to support both a wide range of applications and a wide range of underlying technologies. If a network is either designed with a specific application in mind or designed to be built upon a particular technology, it may perform better for that one case. However, it is unlikely that a unique network can be designed and built for each specific case. Unless there is a reason that a general-purpose solution can't meet your needs, it is better to design a network that can handle a broad spectrum of applications and underlying technologies.

2. **Scalability:**

   The ideal network design would work well with very large networks and also be efficient with small networks. In the past, a network having thousands of nodes might have been considered "large." Now, in standardizing any design, we should be thinking in terms of its operating well with millions or even trillions of nodes. Ideally, efficiency would not be sacrificed if the same design were used on a very small network (say, 20 nodes), but meeting such a goal is unlikely. In such a case, a network designed for very few nodes could be more efficient, using, for example, smaller addresses. But we are willing to compromise somewhat to get a complete answer, provided that it is efficient enough in the special case.

3. **Robustness:**

   Some aspects of robustness are obvious, and most network designs take this into account. For example, the network should continue to operate even if nodes or links fail. Most networks have routing algorithms that adapt to changing topology. However, robustness also has certain more subtle aspects.

   Most networks will work properly in a theoretical world in which no undetected data corruption occurs, all nodes properly execute all the algorithms, parameter settings are compatible across all nodes, and all nodes have sufficient processor power to execute all necessary algorithms in a timely fashion.

   However, we do not live in a theoretical world. Undetected data corruption happens as a result of undetected data errors in transmission or in a node's memory, or during transfer of data internally across a bus. Defective implementations attach to the network. Hardware faults can cause unpredictable behavior. Implementations run out of memory or CPU and behave unpredictably instead of immediately ceasing operation or doing something compatible with the continued effective functioning of the network. Humans (notoriously unreliable components of a system) misconfigure things.

   So robustness in the sense of computing alternative routes is not sufficient. A network should also have the following types of robustness.

   a. **Safety barriers:**

      With most networks, malfunctions can cause widespread disruption. Some networks, however, are designed so that a fault does not spread beyond a safety barrier, and therefore a disruption affects only a portion of the network.

      For example, LAN broadcast storms have been an annoyance with the TCP/IP (Transmission Control Protocol/Internet Protocol) network layer protocol. A *broadcast storm* is an event in which severe congestion is initiated; it is usually caused by bugs in implementations, ambiguous protocol specifications, or misconfigurations. The broadcast storms with IP can

incapacitate a LAN. When two LANs are connected with a bridge, the bridge merges the LANs, and a broadcast storm on either LAN will incapacitate both LANs. If the LANs are instead connected with a router, the broadcast storm will be confined to the single LAN on which it started. Thus, the router acts as a safety barrier through which the LAN broadcast storm does not spread.

Another type of safety barrier is to design a routing protocol hierarchy in which the network is partitioned into *areas* or *domains.* Sometimes the routing protocol can be designed so that disruption in one piece, although it might disable that piece, does not spread to other pieces.

b. **Self-stabilization:**

This concept means that after any sort of database corruption—due to such causes as malfunctioning hardware or undetected data errors—the network will return to normal operation without human intervention within a reasonable time, provided that the faulty hardware is disconnected from the network or repaired and no further data corruption occurs (for some time). Without this type of robustness, an error can cause the network to remain inoperative until every node in the network is simultaneously brought down and rebooted. As you will see in Chapter 12 when routing algorithms are discussed, the routing protocol implemented in the ARPANET (Advanced Research Projects Agency Network) was not self-stabilizing.

This type of robustness does not guarantee that a network will operate properly with a malfunctioning piece of equipment attached, but it makes the network easy to repair after the problem is diagnosed. You need only remove the offending device. Many pieces of equipment get into *wedged states,* and power-cycling them usually works as an instant repair. However, a network does not have an on/off switch and cannot easily be power-cycled. The network's very robustness, in the sense of its being distributed so that it can remain operational even if some parts of it are down, means that if the system is not self-stabilizing, all of it must be "killed" to eliminate any residues of a fault.

If a network is not self-stabilizing, a saboteur can inject a few bad packets and the network will remain down forever or until complex and costly human intervention occurs. If the network is self-stabilizing, a saboteur must inject bad data continually in order to keep the network disrupted. That is far riskier than surreptitiously connecting at some off-hour, injecting a few packets, and quietly slipping away.

Again, if the network is self-stabilizing, repair is simple. After the offending equipment is found, you simply remove it to restore the network to operational status.

c. **Fault detection:**

Although none of today's networks will operate properly in the face of actively malfunctioning nodes (Byzantine failures, discussed next), it would be desirable for a network to have the ability to diagnose itself so that a faulty piece of equipment could be identified. All networks have some ability to detect faults, but none has a perfect ability to do so, and networks vary greatly in the degree to which faults can be identified.

d. **Byzantine robustness:**

The term *Byzantine failure* is taken from a famous problem in computer science known as the *Byzantine generals problem.* A Byzantine failure is one in which a node fails not by simply ceasing operation but instead by acting improperly. Such failure can occur because of defective implementations, hardware faults, or active sabotage. A network with Byzantine robustness would be able to continue working properly even if some portion of the nodes had Byzantine failures. Although none of today's networks has this form of robustness, such networks are possible (see Chapter 16).

4. **Autoconfigurability:**

   Some network designs work well provided that very smart people do a lot of complex management and constantly tweak parameters. Such network designs greatly enhance the job security of the people who understand how to manage them. However, networks of that sort will not suffice in the future. Networks will become too large, they will be divided so that portions are managed by different organizations, and people will be too dependent on networks to rely on a very few experts to keep them running.

   Tomorrow's networks must "run themselves" as much as possible. Ideally, naive users should be able to buy a piece of equipment from the local discount department store, plug it in to a network, and have an operational network. They should not have to configure complex parameters. They should not need to find the address guru to be given an address. (The address guru will be on vacation or will eventually quit, and the envelope on which the address guru scrawled the address assignments will be lost.) Users should not have to find the manager of other nodes to get information about their new node configured into databases.

5. **Tweakability:**

   Networks should come with reasonable defaults and should ideally be autoconfiguring. However, they should also come with timers and other parameters that adventurous network managers can play with to optimize performance for specific conditions. (Ideally, any setting of the parameters will result in reasonable, if not optimal, performance, so even overly adventurous network managers will not be able to inflict much damage.)

6. **Determinism:**

   According to the property of determinism, identical conditions will yield identical results. For example, in a deterministic network design, routes would always be identical given identical physical topologies. In contrast, in a network design that is not deterministic, routes might differ depending on the order in which nodes were brought up in the network.

   Not all people feel that determinism is worth the price, which in some cases means disruptions if the highest-priority element keeps crashing and rebooting. But determinism advocates argue that by ensuring reproducible conditions, determinism makes network analysis much easier.

7. **Migration:**

   A network design will not last forever. It is therefore important to design network protocols so that new features can be added to nodes, one at a time, without disrupting current operations. It is also important to have a design that lets you make modifications, such as address changes, in a node-by-node fashion without disrupting network operations.

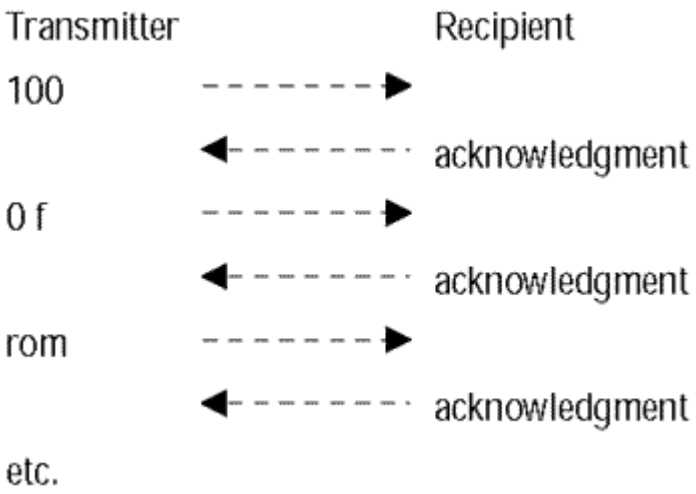
1.4 Reliable Data Transfer Protocols

All data link and transport protocols that provide reliable service tend to have the same general structure. This section introduces the basic ideas and brings up some of the deeper issues involved—issues that are explored in more detail later in the book.

The protocol must deliver a sequence of packets in the same sequence as was transmitted by the source. The protocol has failed if any packets are lost, damaged, misordered, or duplicated. The basic idea is that a packet is transmitted, and the recipient acknowledges its receipt. The packet has a checksum so that the recipient can detect (with high probability) whether the packet was damaged in transit.

In the overly simplified scheme, the transmitter sends a packet, waits for an acknowledgment (also known as an *ack*), and then transmits the next packet (see Figure 1.5). Let's assume that the data being transmitted is the message "1000 from acct A to acct B." Let's further assume that only three characters fit into each packet. If an acknowledgment does not arrive, the transmitter must retransmit the data. Because the transmitter has no way of knowing for sure whether an acknowledgment will arrive, the only thing it can do is to set a timer and assume that if the acknowledgment hasn't arrived within that time, the packet (or the acknowledgment) was probably lost.

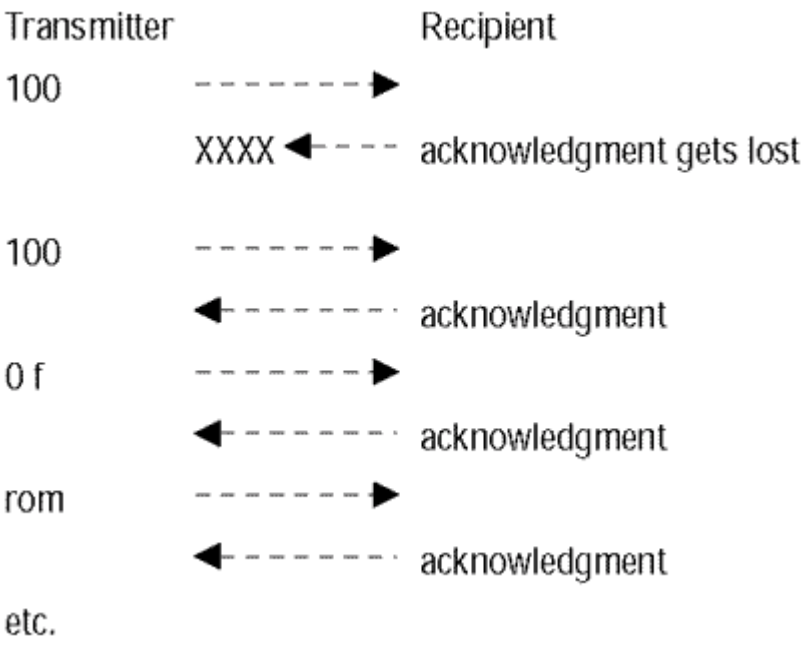
Figure 1.5. Simple protocol, no errors



Establishing the value of the timer is tricky. If the recipient is a busy server, its response time could be highly variable. When the system is heavily loaded, it might take so long to generate an ack that the transmitter will have given up and retransmitted. If the "link" over which the data is being transmitted is a computer network, some packets might take different paths, with the paths having different delays. Or the network might at times be congested (it takes longer to drive the same route at rush hour than at 3:00 a.m. in the U.S. highway network).

If the timer value is too small, packets will be needlessly retransmitted, adding to congestion in the network or adding processing burden to the recipient, which was already too overloaded to return an ack in time (see Figure 1.6). If the timer value is too large, throughput will be delayed after packet loss because the timer will need to expire before further progress can be made on data transmission.

Figure 1.6. Simple protocol, lost ack

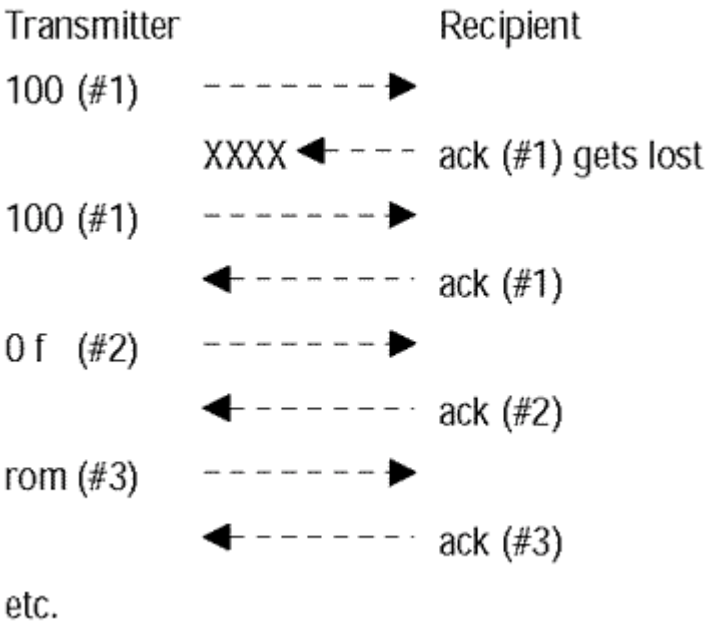


At this point, perhaps a more serious problem than optimizing the timer is that the protocol does not work. If an acknowledgment is lost or delayed, the recipient will get two copies of a packet and will not know that they are duplicates.

The result is that the recipient will assume the message was "1001000 from acct A to acct B." The owner of account B might be delighted to receive more than a million dollars instead of a thousand, but the owner of account A might object. Thus, the protocol must be modified somewhat.

The solution is to add packet numbers, and corresponding numbers in the acknowledgments, so that an ack can be matched with the packet being ack'ed (see Figure 1.7).

Figure 1.7. Adding packet numbers



The recipient receives 100 (#1) twice but, because both packets are marked with (#1), knows that they are duplicates and keeps only one.

Often, data is being transferred simultaneously in both directions. In this case, the packet numbers from right to left are totally independent of the packet numbers from left to right.

There is no ambiguity. An ack number on an ack transmitted from right to left pertains to the stream of numbered packets being transmitted from left to right. An ack number on an ack transmitted from left to right pertains to the stream of numbered packets being transmitted from right to left.

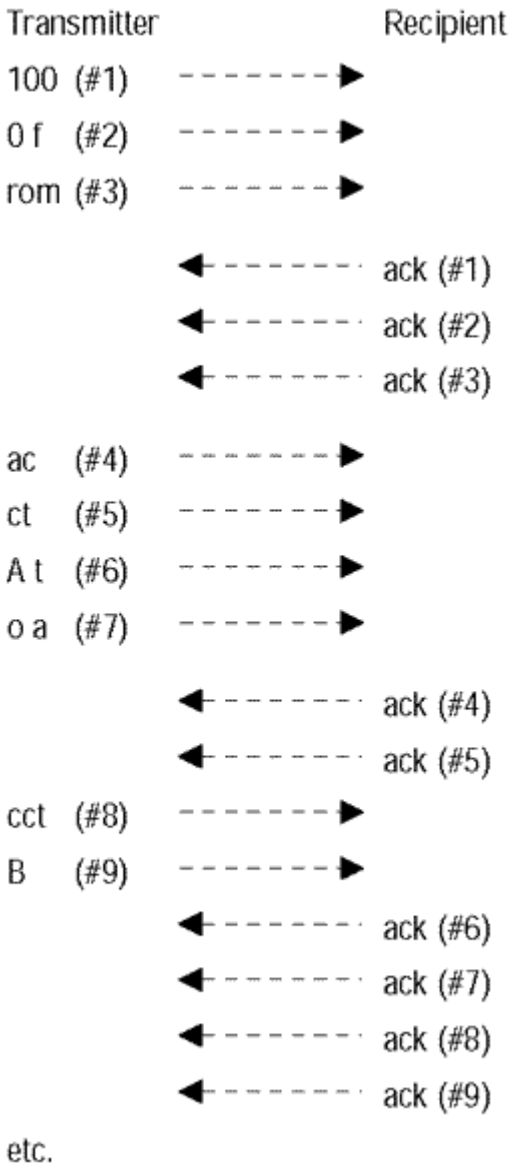
If the transmitter had to wait for an acknowledgment after sending each packet, throughput would be needlessly low. Before the data transmitter can receive an ack, three things must happen after it finishes transmitting a packet. The packet must travel the route between the transmitter and receiver; the receiver must process the packet and generate an acknowledgment; and the acknowledgment must travel the route between the receiver and the transmitter. Because time is available after the transmitter finishes transmitting the packet and before the transmitter finishes processing the acknowledgment, it would be nice to use that time for transmitting more data.

Sending additional data before receiving an acknowledgment for earlier data is known as *pipelining.* The number of packets the transmitter is allowed to have "outstanding" (sent without yet having received an ack) is known as the *window.*

Another issue is the size of the packet number. Ideally the number would be of unlimited size. The first packet of a conversation would be numbered 1, and the 12 billionth would be numbered 12000000000. This would make the protocol very simple. The transmitter could transmit all the packets as quickly as it could. The recipient would then ack all the ones it received, and the transmitter could fill in the holes (noting that acks were not received for, say, packets numbered 17, 112, and 3178).

However, protocols are usually designed to use as few bits as possible for information other than data. (If that were not the case, a government agency responsible for truth in labeling might demand that instead of being called "datagram," the service be called "headergram.") Normally, a limited number of bits is set aside for the packet number; in this example (see Figure 1.8) we'll use 3 bits, although in real protocols a larger number would be desirable.

Figure 1.8. Adding pipelining



With a 3-bit packet number, what happens after packet 7 is transmitted? The answer is that the packet number *wraps around.* Packets are numbered 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4. . . . These types of protocols are usually designed so that an ack need not be transmitted for every packet. Instead, each ack is cumulative in the sense that an ack of packet 4 confirms that all packets through 4 have been received properly.
The finite packet number, together with pipelining and cumulative acks, can create problems unless implemented carefully. Following are some examples.

1. The transmitter transmits 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3. Assume that all packets after the first four are lost, so the recipient gets 0, 1, 2, 3 and returns ack (#3). The transmitter and receiver will assume that all packets arrived properly.
2. The transmitter transmits 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, which are permuted into random order by a network that offers a datagram service. The recipient will have no way of knowing which packet 2 comes first and which packets may be duplicates.
3. The transmitter transmits packets 0, 1, 2, 3, 4, 5, 6, 7. The recipient receives them all and returns ack (#7), which gets lost in transit. The transmitter retransmits all eight packets, which the recipient accepts as new data and returns another ack (#7).

The solution to these problems is to design timers and window sizes carefully. It is important that the packet-numbering space be large enough so that the packet number cannot wrap around during the worst-case delay time. Thus, if it is conceivable that a network could delay a packet by 15 sec, it must not be possible for a transmitter, at maximum speed, to transmit enough packets so that a packet number will wrap around within 15 sec. If the recipient holds on to packets that arrive out of order (for example, if the recipient receives 1 and then 3, it holds on to 3 hoping that 2 will eventually arrive), it is also important that the window be no larger than half the packet number. On strictly sequential channels, packets might be lost but never reordered. If the recipient is guaranteed to discard any packet except the one with the next consecutive number, the window size can be as large as the packet number size minus 1.

Homework

1. Suppose that the session layer hands information down to the transport layer, which transmits that information (plus perhaps some control information) in a single packet. In ISO-ese, this would be stated, "A single SPDU results in a single TSDU, which results in a single TPDU."

   Suppose instead that the session layer gives the transport layer some information that is too large for the transport layer to send. The transport layer instead transmits it as a set of packets, with enough control information so that the transport layer at the destination can reassemble the session layer's information. How would this be expressed in ISO-ese?

2. Now suppose that the transport layer is capable, for efficiency reasons, of taking many little pieces of information handed down by the session layer and putting them all into one big "box" to be transmitted as a single packet. In this way, the transport layer at the destination can sort them back into individual pieces of information. Translate this situation into ISO-ese.
3. Now suppose that the network layer has a packet to send, but some intermediate router notices that the packet is too large to fit over the link on which it should be transmitted. The network layer at the intermediate node breaks the packet into smaller chunks so that they can be reassembled at the destination network layer. Translate this situation into ISO-ese.
4. Assume a packet number size of n and assume a sequential channel. Give an example of a protocol failure in which the transmitter has a window size of n and the receiver discards packets received out of order. Prove that no problems will occur if the window size is n–1 (assume that at start-up all old packets are removed from the channel).
5. Now assume that the receiver does not discard packets received out of order. In other words, the receiver holds on to packet number n, hoping that it will eventually receive packet n–1, and, when it does, it acknowledges them both. Give an example of a protocol failure in which the transmitter has a window size of $n/2+1$. Prove that no problems will occur if the window size is $n/2$.
6. Discuss the trade-offs of providing reliable versus datagram service at the data link layer. Assume that the transport layer will provide reliable connection-oriented service in either case.

   Consider the following points:

   a. The probability of a packet's making it across a sequence of links (which depends on the error rates of the links)
   b. The total number of packet hops required to successfully get a packet to the destination node and get an acknowledgment back to the source
   c. The desire to maximize throughput for the transport layer
   d. The need for the transport layer to estimate the round-trip delay in order to decide when to retransmit a packet that has not been acknowledged

Chapter 2. Data Link Layer Issues

This chapter discusses data link layer issues that affect bridges and routers. One issue is whether service provided by the data link layer should be reliable or datagram-oriented. Another is how to have multiple network layer protocols coexist on a link. When a node receives a packet, how can it tell which protocol suite originated the packet? Although the chapter also discusses many aspects of LAN technology, a fascinating topic, it is not meant to be a detailed reference on LANs. Rather, it explains those aspects of LANs in general, or of specific LAN technologies, that affect the bridging and network layer protocols.

2.1 Generic LANs

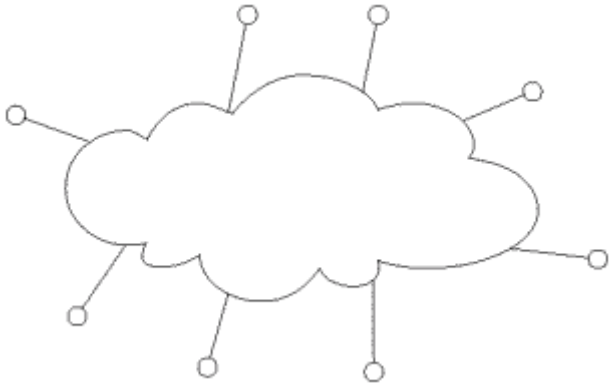
2.1.1 What Is a LAN?

When people use the term LAN, they may refer to any of a number of technologies that have the properties usually associated with LANs. Following are some of those properties.

- Multiple systems attached to a shared medium.
- "High" total bandwidth (the total bandwidth is shared by all the stations).
- "Low" delay.
- "Low" error rate.
- Broadcast capability, also known as multicast capability (the ability to transmit a single message and have it received by multiple recipients).
- Limited geography (several kilometers).
- Limited numbers of stations (hundreds).
- Peer relationship among attached stations (as opposed to a group of slaves with a master). In a peer relationship, all attached stations are equivalent. In a master/slave relationship, one special station, called the *master,* polls the *slaves,* giving each one a turn to transmit.
- Being confined to private property and not subject to PTT (a common abbreviation for Post, Telegraph, and Telephone, a government agency in many countries) regulation.

Note that the meaning of terms such as *low, high,* and *limited* is relative and changes with time. For the purposes of this book, we do not need a definition of a LAN that would distinguish it from a MAN (metropolitan area network) or a WAN (wide area network). (This is fortunate because as LAN technology improves to expanded geographies and WAN technology improves to increased bandwidth, the distinction between LANs and WANs becomes even less clear, and trying to figure out where a MAN fits in is hopeless.) Basically, a LAN (as well as a WAN) can be viewed as a "cloud" to which stations can be attached (see Figure 2.1). If a station attaches to the cloud, it can transmit packets to, and receive packets from, every other station attached to the cloud.

Figure 2.1. Network cloud



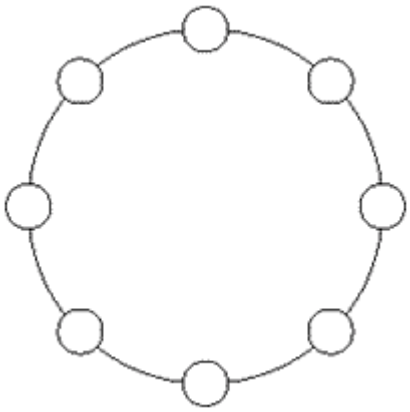
2.1.2 Taking Turns

In a shared medium, only one station can successfully transmit at a time. Some mechanism must exist to allocate bandwidth among the stations so that

- Each station gets a fair share of bandwidth
- Each station gains access to the medium within a reasonable time
- The waste of bandwidth due to arbitration mechanisms is minimized

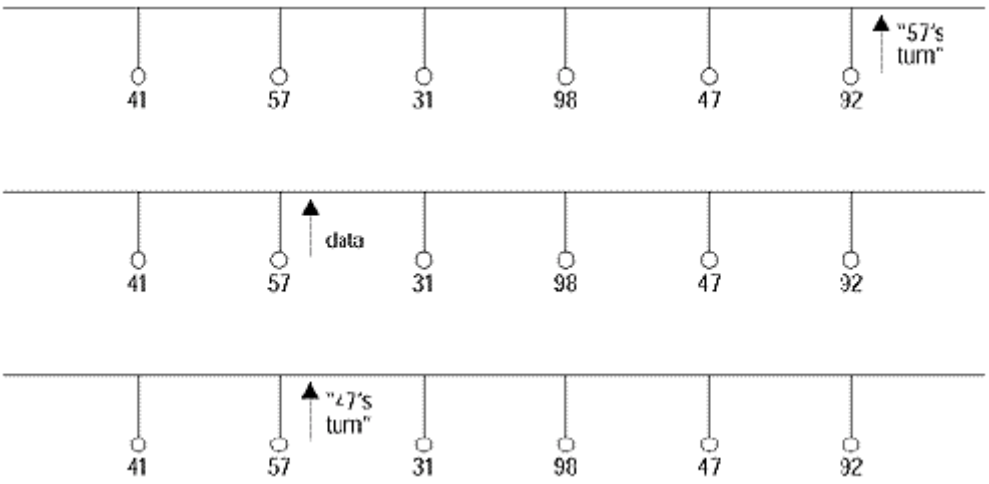
The two most popular bandwidth-arbitration mechanisms used on LANs are token schemes and contention.
In a token scheme, each station is granted permission to transmit in some round-robin fashion. In the case of a *token ring,* a particular sequence of bits known as a *token* travels around the ring (see Figure 2.2). A station is allowed to transmit when it sees the token.

Figure 2.2. Token ring



In the case of a *token bus,* the token is a special packet that is sent from station to station (see Figure 2.3). Each station is required to know the identity of the station to which it should transmit the token.

Figure 2.3. Token bus



In a *contention* scheme, stations transmit at will, and if two stations transmit simultaneously there is a *collision* and neither station's transmission is successful. Mechanisms are built in to minimize the probability of collisions. Contention schemes are *probabilistically fair .* This means that theoretically, some station might never succeed in transmitting because whenever it tries, some other station transmits at the same time. However, contention LANs are carefully designed to make this situation highly unlikely.

Relying on probability instead of a scheme that guarantees a desired outcome might seem worrisome, but you must realize that we rely on probability every day. It is only probably true that enough oxygen atoms will remain in the room in which you are sitting to enable you to continue breathing. It is theoretically possible that through random motion, all oxygen atoms will leave the room and you will asphyxiate.

Even schemes that seem to provide guarantees are actually probabilistic. The token is merely a sequence of bits, which can become corrupted. In a token scheme, it is theoretically possible that one particular station will never succeed in transmitting because the token always gets lost before that station gains access. Or it is possible that every time a station succeeds in acquiring the token, its transmitted packet will get corrupted.

2.2 IEEE 802 LANs

IEEE has a committee known as "802" whose purpose is to standardize LANs. It has standardized not one but several LANs.

LAN protocols cover the bottom two layers of the OSI Reference Model (the physical and data link layers). The 802 committee has chosen to subdivide the data link layer into two sublayers.

1. MAC, which stands for *medium access control* , addresses issues specific to a particular type of LAN. For example, it deals with channel management algorithms such as token passing (802.5 and 802.4), binary backoff after collision detection (802.3), priorities (802.5 and 802.4), error detection, and framing.
2. LLC, which stands for *logical link control,* defines the fields that allow multiple higher-layer protocols to share the use of the data link. Because some people thought that it was not enough for the data link layer to provide a simple datagram service, they pushed for additional functionality. The decision was to provide several flavors of LLC. The following two types are in wide use.
    a. *LLC type 1* is simply a datagram protocol, meaning that a packet is delivered with *best-effort* service by the data link layer. There is no protocol at the data link layer to alert the source as to whether the packet was successfully received. Instead, error control of that sort, if needed, is assumed to be carried out at a higher layer. Note that LLC type 1 doesn't actually do anything because the LAN already gives best-effort service. (That's not a criticism of LLC type 1. It's better to have a protocol that does nothing than a protocol that does something you don't need. Even better would be not to

have a protocol at all when one is not needed. It would make the specs easier to read and would save paper.)

b. *LLC type 2* is a reliable connection-oriented protocol on top of the basic datagram. This means that in addition to the fields required by LLC type 1, there are fields to number packets, provide a piggybacked acknowledgment field, and provide for differentiating data packets from control packets such as acknowledgments and resynchronization messages. LLC type 2 is basically running the connection-oriented data link protocol HDLC (high-level data link control), which was designed for point-to-point links, on top of the LAN datagram-oriented protocol.

The other types of LLC attempt to provide extra reliability without the overhead of LLC type 2. To demystify what the different types of LLC might provide, this book describes LLC types 1 and 2 in detail, but the details of all the types of LLC are not really relevant to bridges and routers.
The IEEE 802 committees relevant to this book are as follows.

- **802.1:**

  This committee deals with issues common across all 802 LANs, including addressing, management, and bridges.

- **802.2:**

  This committee defines LLC. MAC and physical layers are defined for a specific type of LAN by the committee that defines that type of LAN.

- **802.3:**

  This committee deals with the CSMA/CD (carrier sense multiple access with collision detection) LAN. This is derived from the Ethernet, which was invented by Xerox and developed by Digital, Intel, and Xerox.

- **802.4:**

  This committee deals with the token bus LAN.

- **802.5:**

  This committee deals with the token ring LAN.

Other 802 committees deal with such issues as metropolitan area networks and security. Another type of LAN is FDDI (fiber distributed data interface), which is a 100Mb token ring. It is not simply a faster version of 802.5 but rather is very different from 802.5. (Although the technical differences between them are fascinating, they are not relevant to this book.) FDDI was standardized by ANSI rather than IEEE.

2.3 Names, Addresses, Routes

Considerable confusion exists regarding the terms *name, address,* and *route.* Shoch[1] defines these terms as follows.

[1] J. Shoch, "Internetwork Naming, Addressing, and Routing," *Compcon* (Fall 1978), 72–79.

- *Name:* what something is
- *Address:* where it is

- *Route:* how to get there

The Shoch paper seems to be referenced whenever any mention is made of names, addresses, or routes. However, I have never found these definitions helpful, in the sense of enabling one to look at a string of bits and decide, based on the preceding taxonomy, whether the string should be classified as a name, an address, or a route.
A helpful alternative method of defining the three concepts is as follows. Suppose a particular string of bits refers to a particular station. We want to decide whether that string of bits should be considered a name, an address, or a route. In the following definitions, the *destination* is the station referred to by the string of bits, and the *source* is the station that is using the string of bits to refer to the destination.

1. **Name:**

   A name is location-independent with respect to both the source and the destination. If something is the name of a destination, it will remain unchanged even if the destination moves, and it is valid regardless of which source is attempting to reach the destination. An example of a name is a Social Security number, which remains unchanged even if the number's owner moves. Sometimes, fields that are names are referred to as *identifiers,* or *IDs.*

2. **Address:**

   An address is valid regardless of the location of the source station, but it may change if the destination moves. An example of an address is a postal address. The same destination postal address works regardless of the location from which a letter is mailed. However, if the destination moves, it is assigned a new address.

3. **Route:**

   A route is dependent on the location of both the source and the destination. In other words, if two sources specify a route to a given destination, the routes are likely to differ. And if the destination moves, all routes to it are likely to change. An example of a route is the statement "To get to my house, go west three miles and take a right turn at the first light. It's the last house on the left."

Note that with the preceding descriptions, the entities known as "addresses" in 802 would be classified as "names" rather than "addresses." Especially with globally assigned 48-bit "LAN addresses," the LAN address will not change if a station moves to a different LAN. There are probably several reasons that these fields are referred to as addresses.

- Some people like to refer to something as a "name" if it is human-friendly—an ASCII string rather than a bunch of bits. Because 48-bit quantities are certainly unpleasant to type, remember, or look at, those who equate the word *name* with human compatibility prefer to refer to them as addresses. I would rather refer to a human-hostile quantity that is location-independent as an identifier, or ID.
- From the viewpoint of a higher-layer process that may move from node to node, its "LAN address" actually becomes an address because when the higher-layer process moves to a different node, its LAN address changes.
- If the 48-bit "LAN address" is stored in the interface to the LAN rather than in the node, then a node with multiple attachments to LANs has multiple LAN addresses. But then the 48-bit quantity is addressing not the node but rather one of the interfaces of the node, and I would therefore claim that the 48-bit quantity is an identifier of the interface.

Because everyone in the industry refers to the 48-bit elements as "addresses," I do so in this book. It is important to realize, however, that most terms (*layer, address, route, node,*

network, LAN, and so on) are at best vaguely defined in the industry and are used in conflicting ways by various communities.

2.4 LAN Addresses

In many LAN technologies, every station on a LAN hears every packet transmission, so it is necessary to include a **destination** field in each packet. So that the destination can identify which station transmitted the packet, a **source** field is also included. To prevent software interrupts with every packet, LAN adapters can filter out packets not addressed to the station. The 802 committee needed to standardize addresses for its LANs. The first decision was to set the length of the address field. The committee apparently thought that if standardizing on one size was a good thing, standardizing on several sizes would be even better. The 802 committee gave the option of running a LAN (other than 802.6) with 48-bit addresses or 16-bit addresses. It gave the option of running 802.6 with 16-, 48-, or 60-bit addresses. Luckily, 16-bit addresses have not caught on and can safely be ignored.

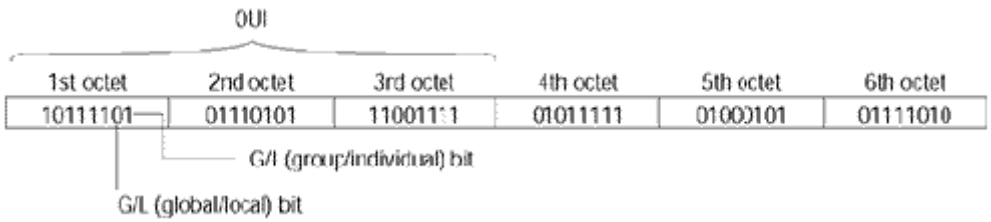
The argument for 16-bit addresses is that this size is sufficient for any single LAN provided that the manager of the LAN is capable of assigning addresses to the stations. Also, 802.4 used addresses to resolve an initial contention phase prior to building a logical ring and would come up or restart faster with 16-bit addresses.

The argument for 48-bit addresses is that they enable stations to be provided with a globally unique identifier at the time of manufacture. This allows networks to be truly *plug and play,* in the sense that a customer could buy an off-the-shelf system, plug it in to the network, and have it operate, without having to first assign it an address.

The way globally unique addresses work is that a global authority is responsible for handing out blocks of addresses. Originally, Xerox was the global authority; now the official global authority is IEEE. When a vendor wishes to manufacture equipment that will plug in to a LAN, it first contacts the global authority to obtain a block of addresses. The current cost of a block of addresses is $1,250, for which the vendor is given 2^{24} addresses. In other words, the vendor is given three fixed-value octets, with the remaining three octets being for the vendor to allocate. The fixed-value portion of the address is sometimes referred to in the industry as the *vendor code* or OUI (*organizationally unique identifier*), but that is really a misnomer (misaddresser?) because a vendor can purchase more than one block of addresses as well as donate addresses to other vendors.

The three fixed-value octets actually have additional structure (see Figure 2.4). One bit represents *group/individual.* If that bit is 0, the address refers to a particular station; if that bit is 1, the address refers to a logical group of stations. Thus, the global authority does not really give 24 bits of fixed value but rather gives 23 bits of fixed value, with the remaining bit being the group/individual bit. An entire address is 6 octets long. The 3 octets of constant leave an additional 3 octets (24 bits) that can be assigned by the vendor. Therefore, when a vendor purchases a block of addresses, it gets 2^{24} station addresses and 2^{24} group addresses.

Figure 2.4. IEEE address



The 802 committee was not sure that everyone would want to go to the trouble (and expense) of obtaining a block of addresses from the global authority. Therefore, it designated another of the 48 bits to indicate whether the address was globally or locally assigned. If a vendor purchases a block of addresses from the global authority, the global/local bit will be set to 0. People are free to use any addresses with the global/local bit set to 1. However, if local addresses are used, it is up to the network manager to assign addresses and make sure that there are no *address collisions* (which occur when two stations use the same address). Address collision becomes an important issue when two networks are merged.

Group addresses are also sometimes referred to as *multicast addresses.* The most common use of multicast addresses is for discovering appropriate neighbors (nodes on the same link) by one of the following two methods.

1. **Solicitation:**

   Suppose the network contains one or more of a certain type of station—for example, a naming server, a router, or a file server—that station A is likely to want to contact. Management could configure station A, with the addresses of all those stations. However, it would be more desirable if station A did not need to know about specific servers a priori. Instead, it would know a single group address, Zservers (where "Z" can be any type of service such as those just suggested).

   When station A wishes to find a Z server, it transmits a packet with the destination address Zservers. All the Z servers listen for, and respond to, packets directed to that address.

2. **Advertisement:**

   A different way to use group addresses is to define an address to be used for stations listening for a service. Instead of having service Z clients ask for help by transmitting to Zservers, Z servers would periodically transmit packets to the address Zclients. A Z client would listen for packets addressed to Zclients until it heard such a packet. Then, based on the source address of the packet or on some other field explicitly contained in the data portion of the packet, the Z client would now know the address of a Z server.

   The human counterpart of this method is commercial advertising. The advertising industry would love to be able to transmit an advertisement that would be received only by people interested in hearing it, but the best advertisers can do is to advertise in media whose audiences tend to match the type of people who might be interested in a particular product.

2.5 Multicast versus Unicast Addresses

Why is it that a multicast address looks different from an individual address? If a particular address is designated to mean "all Z servers" and if all Z servers are supposed to listen to that address in addition to their own, why can't just any address be used for a multicast?
The problem is that on a LAN there are a lot of packets, and it would seriously degrade the performance of an attached station if the software had to process an interrupt every time a packet for any destination was transmitted on the wire. The hardware will receive every packet, and it is possible to request that it deliver every packet. This is known as listening *promiscuously.*
Sometimes, as in the case of a bridge or a LAN-monitoring tool, it is appropriate for the hardware to deliver every packet, but for most applications, promiscuity is not desirable. Instead, it should be possible for the hardware to look at enough of a packet to decide whether the software might conceivably be interested in the packet. If the packet is of interest, the hardware should pass it up to the software. If it is not, the hardware should just drop the packet (*filter* it).
Theoretically, it is not necessary to reserve a bit in the address to differentiate group and individual addresses. Ideally, the software would tell the chip all the different addresses that the software was interested in receiving, and the chip would pass packets up to the software if and only if the "destination address" field in the data link header matched one of the addresses requested by the software.
The problem with having the software request a certain number of addresses is that the chip designer would need to pick a maximum number of addresses that could be requested. If the designer picked too large a number, the chip would be too expensive. If the designer picked