COMPUTER NETWORKS

M.Sc., CS I year Paper - IX

Lesson Writer

P. AMMI REDDY, ***M.Sc., M.C.A., M.Tech.***
Associate Professor
Vasireddy Venkatadri Institute of Technology
Nambur (P.O.), GUNTUR – Dt.

Editor & Advisor for the Course

Prof. E.SREENIVASA REDDY, M.Tech., Ph.D.
Principal
Vasireddy Venkatadri Institute of Technology
Nambur (P.O.), GUNTUR – Dt.

Director
Prof.V.CHANDRASEKHARA RAO, M.Com., Ph.D.

CENTRE FOR DISTANCE EDUCATION
ACHARAYA NAGARJUNA UNIVERSITY
NAGARJUNA NAGAR – 522 510

Ph: 0863-2293299,2293356,08645-211023,Cell:98482 85518
08645-21102 4 (Study Material)
Website: www.anucde.com, e-mail:anucde@yahoo.com

Contents

UNIT – I

1. Introduction and Course Outline

Structure

Objective

- Define basic functions of the Computer Networks
- State the evolution of Computer Networks
- Categorize different types of Computer Networks
- Specify some of the application of Computer Networks

1.1 Introduction

The concept of Network is not new. In simple terms it means an interconnected set of some objects. For decades we are familiar with the Radio, Television, railway, Highway, Bank and other types of networks. In recent years, the network that is making significant impact in our day-to-day life is the **Computer network**. By computer network we mean an interconnected set of autonomous computers. The term autonomous implies that the computers can function independent of others. However, these computers can exchange information with each other through the communication network system. Computer networks have emerged as a result of the convergence of two technologies of this century- Computer and Communication as shown in Fig. 1.1. The consequence of this revolutionary merger is the emergence of a integrated system that transmit all types of data and information. There is no fundamental difference between data communications and data processing and there are no fundamental differences among data, voice and video communications. After a brief historical background in Section 1.2, Section 1.2 introduces different network categories. A brief overview of the applications of computer networks is presented in

Section 1.3. Finally an outline of the entire course is given in Section 1.4.

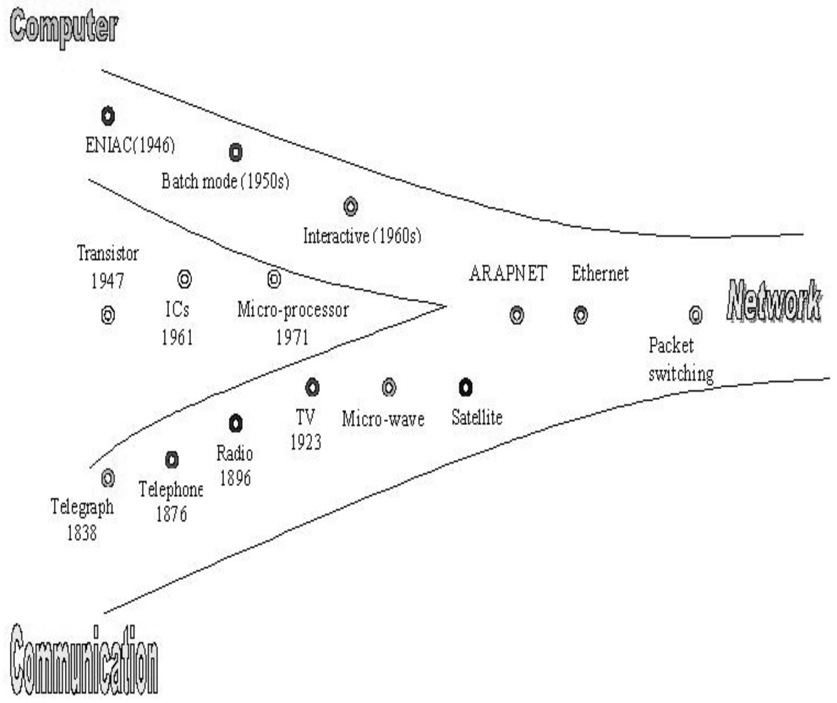


Figure 1.1 *Evolution of computer networks*

1.2 Historical Background

The history of electronic computers is not very old. It came into existence in the early 1950s and during the first two decades of its existence it remained as a centralized system housed in a single large room. In those days the computers were large in size and were operated by trained personnel. To the users it was a remote and mysterious object having no direct communication with the users. Jobs were submitted in the form of punched cards or paper tape and outputs were collected in the form of computer printouts. The submitted jobs were executed by the computer one after the other, which is referred to as batch mode of data processing. In this scenario, there was long delay between the submission of jobs and receipt of the results.

In the 1960s, computer systems were still centralizing, but users provided with direct access through interactive terminals connected by point-to-point low-speed data links with the computer. In this situation, a large number of users, some of them located in remote locations could simultaneously access the centralized computer in time-division multiplexed mode. The users could now get immediate interactive feedback from the computer and correct errors immediately. Following the introduction of on-line terminals and time-sharing operating systems, remote terminals were used to use the central computer.

With the advancement of VLSI technology, and particularly, after the invention of microprocessors in the early 1970s, the computers became smaller in size and less expensive, but with significant increase in processing power. New breed of low-cost computers known as mini and personal computers were introduced. Instead of having a single central computer, an organization could now afford to own a number of computers located in different departments and sections.

Side-by-side, riding on the same VLSI technology the communication technology also advanced leading to the worldwide deployment of telephone network, developed primarily for voice communication. An organization having computers located geographically dispersed locations wanted to have data communications for diverse applications. Communication was required among the machines of the same kind for collaboration, for the use of common software or data or for sharing of some costly resources. This led to the development of computer networks by successful integration and cross-fertilization of communications and geographically dispersed computing facilities. One significant development was the APPANET (Advanced Research Projects Agency Network). Starting with four-node experimental network in 1969, it has subsequently grown into a network several thousand computers spanning half of the globe, from Hawaii to Sweden. Most of the present-day concepts such as packet switching evolved from the ARPANET project. The low bandwidth (3 KHz on a voice grade line) telephone network was the only generally available communication system available for this type of network.

The bandwidth was clearly a problem, and in the late 1970s and early 80s another new communication technique known as Local Area Networks (LANs) evolved, which helped computers to communicate at high speed over a small geographical area. In the later years use of optical fiber and satellite communication allowed high-speed data communications over long distances.

1.3 Network Technologies

There is no generally accepted taxonomy into which all computer networks fit, but two dimensions stand out as important: **Transmission Technology** and **Scale**. The classifications based on these two basic approaches are considered in this section.

1.3.1 Classification Based on Transmission Technology

Computer networks can be broadly categorized into two types based on transmission technologies:

- Broadcast networks
- Point-to-point networks

1.3.1.1 Broadcast Networks

Broadcast network have a single communication channel that is shared by all the machines on the network as shown in Figs.1.2 and 1.3. All the machines on the network receive short messages, called packets in certain contexts, sent by any machine. An address field within the packet specifies the intended recipient. Upon receiving a packet, machine checks the address field. If packet is intended for itself, it processes the packet; if packet is not intended for itself it is simply ignored.

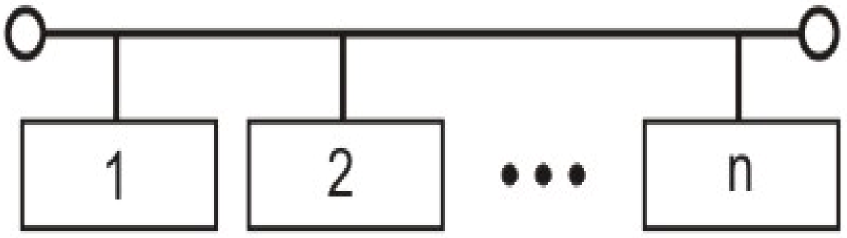


Figure 1.2 *Example of a broadcast network based on shared bus*

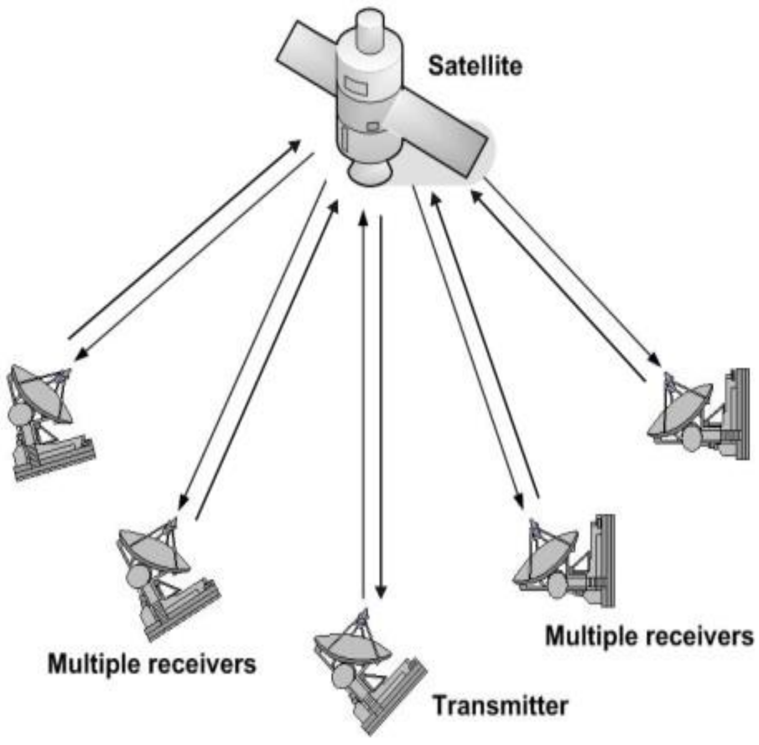


Figure 1.3 *Example of a broadcast network based on satellite communication*

This system generally also allows possibility of addressing the packet to all destinations (all nodes on the network). When such a packet is transmitted and received by all the machines on the network. This mode of operation is known as *Broadcast Mode*. Some Broadcast systems also support transmission to a sub-set of machines, something known as *Multicasting*.

1. 3.1.2 Point-to-Point Networks

A network based on point-to-point communication is shown in Fig. 1.4. The end devices that wish to communicate are called *stations*. The switching devices are called *nodes*. Some Nodes connect to other nodes and some to attached stations. It uses FDM or TDM for node-to-node communication. There may exist multiple paths between a source-destination pair for better network reliability. The switching nodes are not concerned with the contents of data. Their purpose is to provide a switching facility that will move data from node to node until they reach the destination.

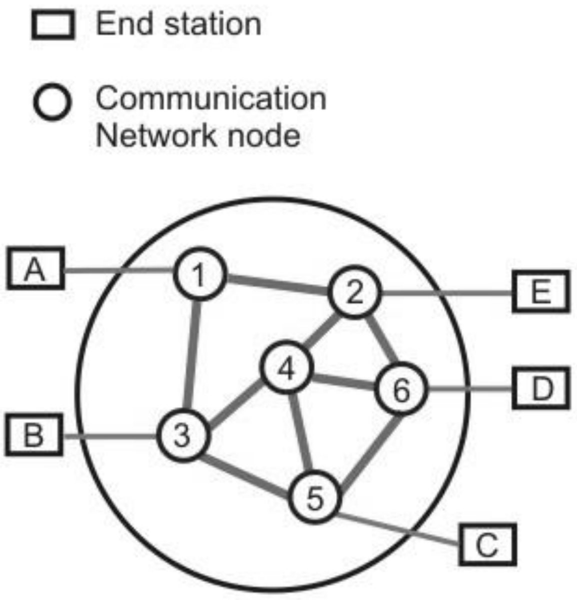


Figure 1.4 *Communication network based on point-to-point communication*

As a general rule (although there are many exceptions), smaller, geographically localized networks tend to use broadcasting, whereas larger networks normally use are point-to- point communication.

1.3.2 Classification based on Scale

Alternative criteria for classifying networks are their scale. They are divided into Local
Area (LAN), Metropolitan Area Network (MAN) and Wide Area Networks (WAN).

1.3.2.1 Local Area Network (LAN)

LAN is usually privately owned and links the devices in a single office, building or campus of up to few kilometers in size. These are used to share resources (may be hardware or software resources) and to exchange information. LANs are distinguished from other kinds of networks by three categories: their size, transmission technology and topology.

LANs are restricted in size, which means that their worst-case transmission time is bounded and known in advance. Hence this is more reliable as compared to MAN and WAN. Knowing this bound makes it possible to use certain kinds of design that would not otherwise be possible. It also simplifies network management.

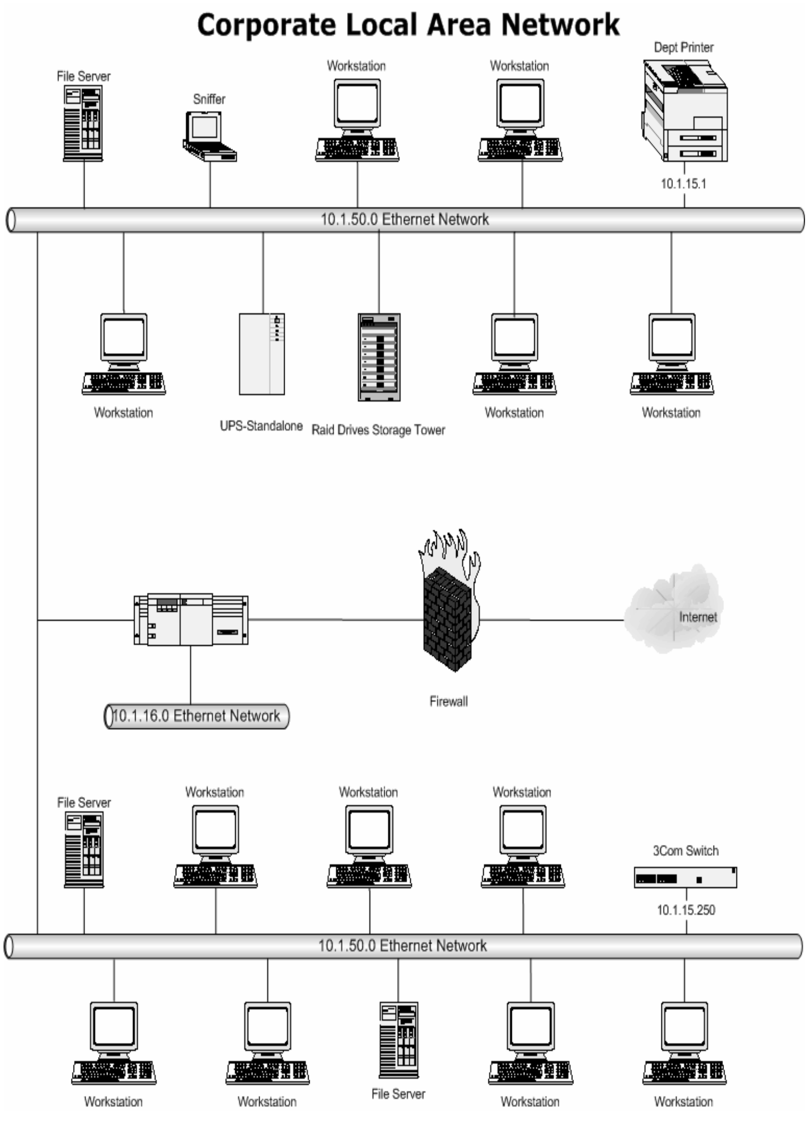


Figure 1.5 Local Area Network

LAN typically used transmission technology consisting of single cable to which all machines are connected. Traditional LANs run at speeds of 10 to 100 Mbps (but now much higher speeds can be achieved). The most common LAN topologies are bus, ring and star. A typical LAN is shown in Fig. 1.5.

1.3.2.2 Metropolitan Area Networks (MAN)

MAN is designed to extend over the entire city. It may be a single network as a cable TV network or it may be means of connecting a number of LANs into a larger network so that resources may be shared as shown in Fig. 1.6. For example, a company can use a MAN to connect the LANs in all its offices in a city. MAN is wholly owned and operated by a private company or may be a service provided by a public company.

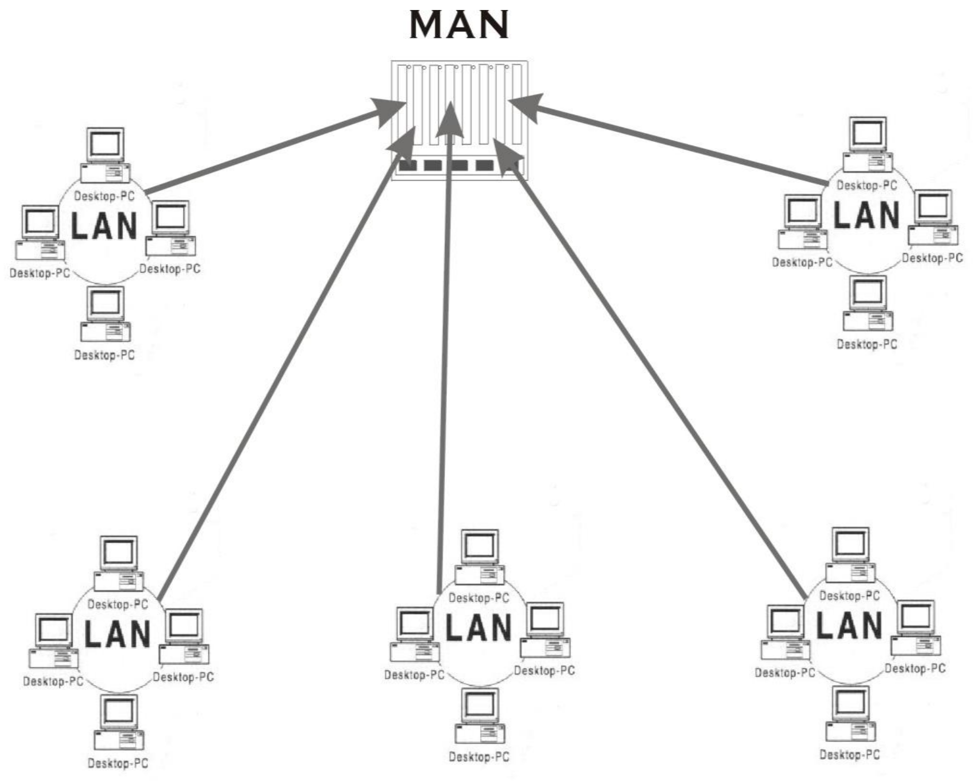


Figure 1.6 Metropolitan Area Networks (MAN)

The main reason for distinguishing MANs as a special category is that a standard has been adopted for them. It is **DQDB** (Distributed Queue Dual Bus) or IEEE 802.6.

1.3.2.3 Wide Area Network (WAN)

WAN provides long-distance transmission of data, voice, image and information over large geographical areas that may comprise a country, continent or even the whole world. In contrast to LANs, WANs may utilize public, leased or private communication devices, usually in combinations, and can therefore span an unlimited number of miles as shown in Fig. 1.7. A WAN that is wholly owned and used by a single company is often referred to as *enterprise network*.

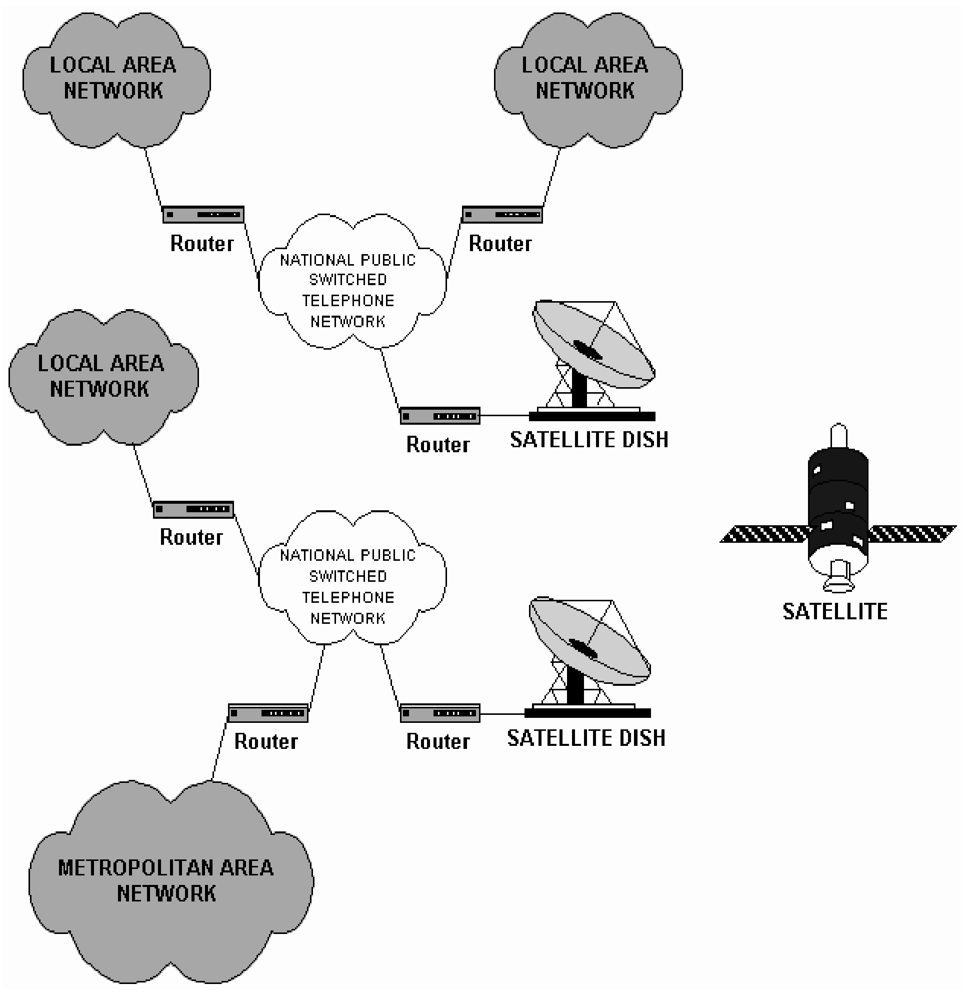


Figure 1.7 *Wide Area Network*

1.3.2.4 The Internet

Internet is a collection of networks or network of networks. Various networks such as LAN and WAN connected through suitable hardware and software to work in a seamless manner. Schematic diagram of the Internet is shown in Fig. 1.8. It allows various applications such as e-mail, file transfer, remote log-in, World Wide Web, Multimedia, etc run across the internet. The basic difference between WAN and Internet is that WAN is owned by a single organization while internet is not so. But with the time the line between WAN and Internet is shrinking, and these terms are sometimes used interchangeably.

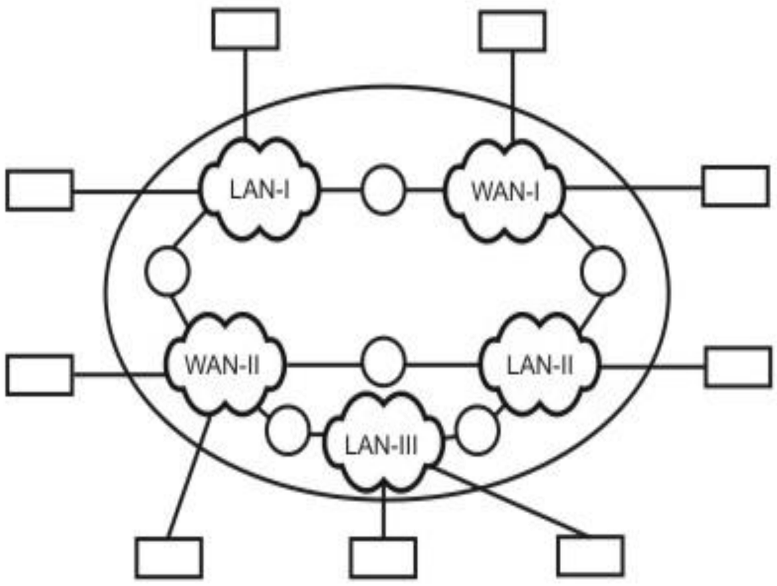


Figure 1.8 Internet – network of networks

1.4 Applications

In a short period of time computer networks have become an indispensable part of business, industry, entertainment as well as a common-man's life. These applications have changed tremendously from time and the motivation for building these networks are all essentially economic and technological.

Initially, computer network was developed for defense purpose, to have a secure communication network that can even withstand a nuclear attack. After a decade or so, companies, in various fields, started using computer networks for keeping track of inventories, monitor productivity, communication between their different branch offices located at different locations. For example, Railways started using computer networks by connecting their nationwide reservation counters to provide the facility of reservation and enquiry from any where across the country.

And now after almost two decades, computer networks have entered a new dimension; they are now an integral part of the society and people. In 1990s, computer network started delivering services to private individuals at home. These services and motivation for using them are quite different. Some of the services are access to remote information, person-person communication, and interactive entertainment. So, some of the applications of computer networks that we can see around us today are as follows:

Marketing and sales:

Computer networks are used extensively in both marketing and sales organizations. Marketing professionals use them to collect, exchange, and analyze data related to customer needs and product development cycles. Sales application includes teleshopping, which uses order-entry computers or telephones connected to order processing network, and online-reservation services for hotels, airlines and so on.

Financial services:

Today's financial services are totally depended on computer networks. Application includes credit history searches, foreign exchange and investment services, and electronic fund transfer, which allow user to transfer money without going into a bank (an automated teller machine is an example of electronic fund transfer, automatic pay-check is another).

Manufacturing:

Computer networks are used in many aspects of manufacturing including manufacturing process itself. Two of them that use network to provide essential services are computer-aided design (CAD) and computer-assisted manufacturing (CAM), both of which allow multiple users to work on a project simultaneously.

Directory services:

Directory services allow list of files to be stored in central location to speed worldwide search operations.

Information services:

A Network information service includes bulletin boards and data banks. A World Wide Web site offering technical specification for a new product is an information service.

Electronic data interchange (EDI):

EDI allows business information, including documents such as purchase orders and invoices, to be transferred without using paper.

Electronic mail:

Probably it's the most widely used computer network application.

Teleconferencing:

Teleconferencing allows conference to occur without the participants being in the same place. Applications include simple text conferencing (where participants communicate through their normal keyboards and monitor) and video conferencing where participants can even see as well as talk to other fellow participants. Different types of equipments are used for video conferencing depending on what quality of the motion you want to capture (whether you want just to see the face of other fellow participants or do you want to see the exact facial expression).

Voice over IP:

Computer networks are also used to provide voice communication. This kind of voice communication is pretty cheap as compared to the normal telephonic conversation.

Video on demand:

Future services provided by the cable television networks may include video on request where a person can request for a particular movie or any clip at anytime he wish to see.

Summary:

The main area of applications can be broadly classified into following categories:

Scientific and Technical Computing

– Client Server Model, Distributed Processing, Parallel Processing, Communication Media

Commercial

– Advertisement, Telemarketing, Teleconferencing
– Worldwide Financial Services

Network for the People (this is the most widely used application nowadays) – Telemedicine, Distance Education, Access to Remote Information, Person-to- Person Communication, Interactive Entertainment.

2. Internet & Protocols

Structure

Objectives

- Description of Internet
- Define protocol
- Understand concept of network edge
- Discuss switching functions
- Have a broad idea about the different transmission impairments

2-1 Introduction

Computer networking is one of the most exciting and important technological fields of our time. The Internet interconnects millions (and soon billions) of computers, providing a global communication, storage, and computation infrastructure. Moreover, the Internet is currently being integrated with mobile and wireless technology, ushering in an impressive array of new applications. Yes, computer networking has indeed come a long way since its infancy in the 1960s. But this is only the beginning a new generation of creative engineers and computer scientists will drive the Internet to yet unforeseen terrains. In this lesson, after introducing some basic terminology and concepts, we will first examine the "edge" of a computer network. We'll look at the end systems and network applications, and the transport services provided to these applications.

2-2 What Is the Internet?

An *internetwork* is a collection of individual networks, connected by intermediate networking devices, that functions as a single large network. Internetworking refers to the industry, products, and procedures that meet the challenge of creating and administering internetworks.

The Internet has revolutionized many aspects of our daily lives. It has affected the way we do business as well as the way we spend our leisure time. The Internet is a communication system that has brought a wealth of information to our fingertips and organized it for our use.

2-2-1 Nuts-and-Bolts Description

Instead of giving a one-sentence definition, let's try a more descriptive approach. There are a couple of ways to do this. One way is to describe the nuts and bolts of the Internet, that is, the basic hardware and software components that make up the Internet. Another way is to describe the Internet in terms of a networking infrastructure that provides services to distributed applications. Let's begin with the nuts-and-bolts description, using Figure 2-1 to illustrate our discussion.

The public Internet is a worldwide computer network, that is, a network that interconnects millions of computing devices throughout the world. Most of these computing devices are traditional desktop PCs, UNIX-based workstations, and so-called servers that store and transmit information such as Web pages and e-mail messages. Increasingly, nontraditional Internet end systems such as PDAs (Personal Digital Assistants), TVs, mobile computers, automobiles, and toasters are being connected to the Internet. In the Internet jargon, all of these devices are called **hosts** or **end** systems.

End systems are connected together by **communication links.** There are many types of communication links, which are made up of different types of physical media, including coaxial cable, copper wire, and fiber optics and radio spectrum. Different links can transmit data at different rates. The link transmission rate is often called the **bandwidth** of the link, which is typically measured in bits/second.

End systems are not usually directly attached to each other via a single communication link. Instead, they are indirectly connected to each other through intermediate switching devices known as **routers.** A router takes a chunk of information arriving on one of its incoming communication links and forwards that chunk of information on one of its outgoing communication links. In the jargon of computer networking, the chunk of information is called a **packet.** The path that the packet takes from the sending end system, through a series of communication links and routers, to the receiving end system is known as a **route** or **path** through the network. Rather than provide a *dedicated* path between communicating end systems, the Internet uses a technique known as **packet switching** that allows multiple communicating end systems to share a path, or parts of a path, at the same time. The first packet-switched networks, created in the 1970s, are the earliest ancestors of today's Internet

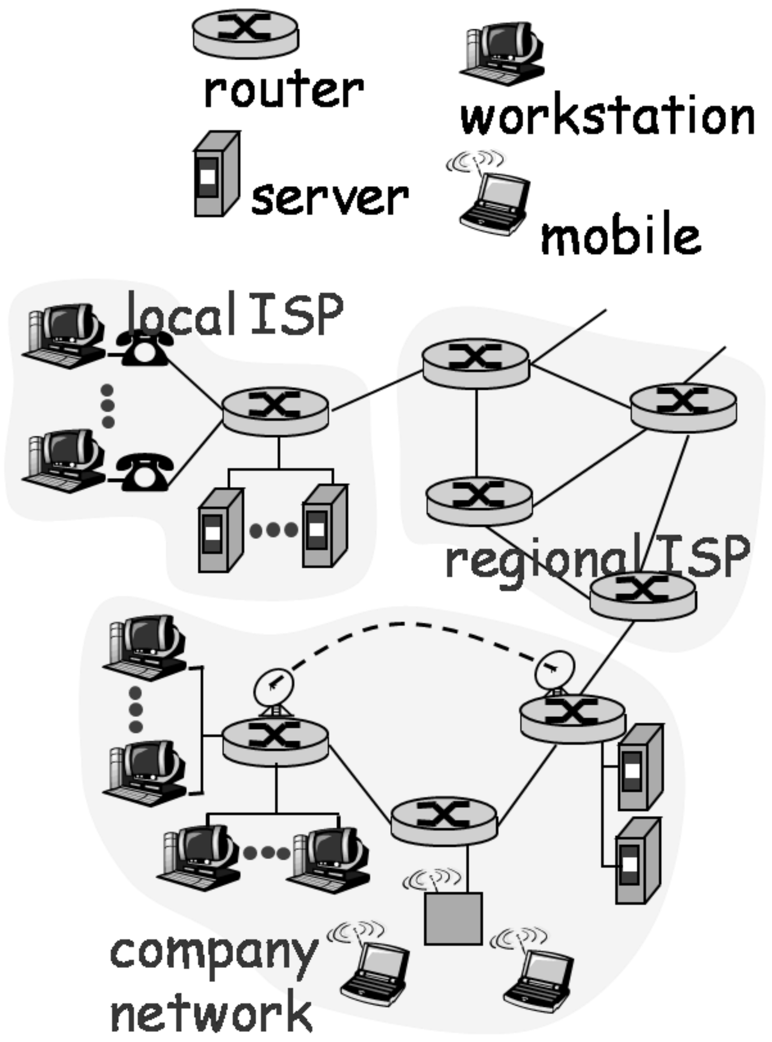


Figure 2-1 some pieces of the Internet

End systems access the Internet through **Internet Service Providers** (ISPs), including residential ISPs such as AOL or MSN, university ISPs such as Stanford University, and corporate ISPs such as Ford Motor Company. Each ISP is a network of routers and communication links. The different ISPs provide a variety of different types of network access to the end systems, including 56 Kbps dial-up modem access, residential broadband access such as cable modem or DSL, high-speed LAN access, and wireless access. ISPs also provide Internet access to content providers, connecting Web sites directly to the Internet.

End systems, routers, and other "pieces" of the Internet, run **protocols** that control the sending and receiving of information within the Internet. **TCP** (the Transmission Control Protocol) and **IP** (the Internet Protocol) are two of the most important protocols in the Internet. The IP protocol specifies the format of the packets that are sent and received among routers and end systems. The Internet's principal protocols are collectively known as **TCP/IP.**

At the technical and developmental level the Internet is made possible through creation, testing, and implementation of **Internet standards.** These standards are developed by the Internet Engineering Task Force (IETF). The IETF standards documents are called **RFCs** (request for comments). RFCs started out as general requests for comments (hence the name) to resolve architecture problems that faced the precursor to the Internet. RFCs, though not formally standards, have evolved to the point where they are cited as such. RPCs tend to be quite technical and detailed. They define protocols such as TCP, IP, HTTP (for the Web), and SMTP (for open-standards e-mail). There are more than 3,000 different RFCs.

2.2.2 A Service Description

Service-oriented views can be description as

• The Internet allows **distributed applications** running on its end systems to exchange data with each other. These applications include remote login, electronic mail, Web surfing, instant messaging, audio and video streaming. Internet telephony, distributed games, peer-to-peer (P2P) file sharing, and much, much more. It is worth emphasizing that "the Web" is not a separate network but rather just one of many distributed applications that use the communication services provided by the Internet.

• The Internet provides two services to its distributed applications: a **connection-oriented reliable service** and a **connectionless unreliable service.** Loosely speaking, the connection-oriented reliable service guarantees that data transmitted from a sender to a receiver will eventually be delivered to the receiver in order and in its entirety. The connectionless unreliable service does not make any guarantees about eventual delivery. Typically, a distributed application makes use of one or the other (but not both) of these two services.

• Currently, the Internet does not provide a service that makes promises about *how long* it will take to deliver the data from sender to receiver. And except for increasing your access bandwidth to your Internet service provider, you currently cannot obtain better service (for example, bounded delays) by paying more a state of affairs that some (particularly Americans) find odd.

This second description of the Internet that is, in terms of the services it provides to distributed applications is a nontraditional, but important, one. Increasingly, advances in the nuts-and-bolts components of the Internet are being driven by the needs of new applications. So it's important to keep in mind that the Internet is an *infrastructure* in which new applications are being constantly invented and deployed.

We have just given two descriptions of the Internet, one in terms of its hardware and software components, the other in terms of the services it provides to distributed applications.

2.3 What Is a Protocol?

Imagine the number of people communicating in the world, the number of different languages they use, the number of different machines they use, the number of ways in which they transmit data and the different software they use. We would never be able to communicate worldwide if there were no 'standards' governing the way we communicate and the way our machines treat data. These standards are sets of rules.

There are rules governing how data is transferred over networks, how they are compressed, how they are presented on the screen and so on. These set of rules are called protocols. There are many protocols, each one governing the way a certain technology works. For example, the IP protocol defines a set of rules governing the way computers use IP packets to send data over the Internet or any other IP-based network.

2-3-1 A Human Analogy

It is probably easiest to understand the notion of a computer network protocol by first considering some human analogies, since we humans execute protocols all of the time. Consider what you do when you want to ask someone for the time of day. A typical exchange is shown in Figure 2.2. 'Human protocol (or good manners, at least) dictates that one first offer a greeting ^fee first "Hi" in Figure 2.2) to initiate communication with someone else. The typical response to a "Hi" is a returned "Hi" mes-sage. Implicitly, one then takes a cordial "Hi" response as an indication that one can proceed and ask for the time of day. A different response to the initial "Hi" (such as "Don't bother me!" or "I don't speak English," or some unprintable reply) might indicate an unwillingness or inability to communicate. In this case, the human protocol would be not to ask for the time of day. Sometimes one gets no response at all to a question, in which case one typically gives up asking that person for the time.

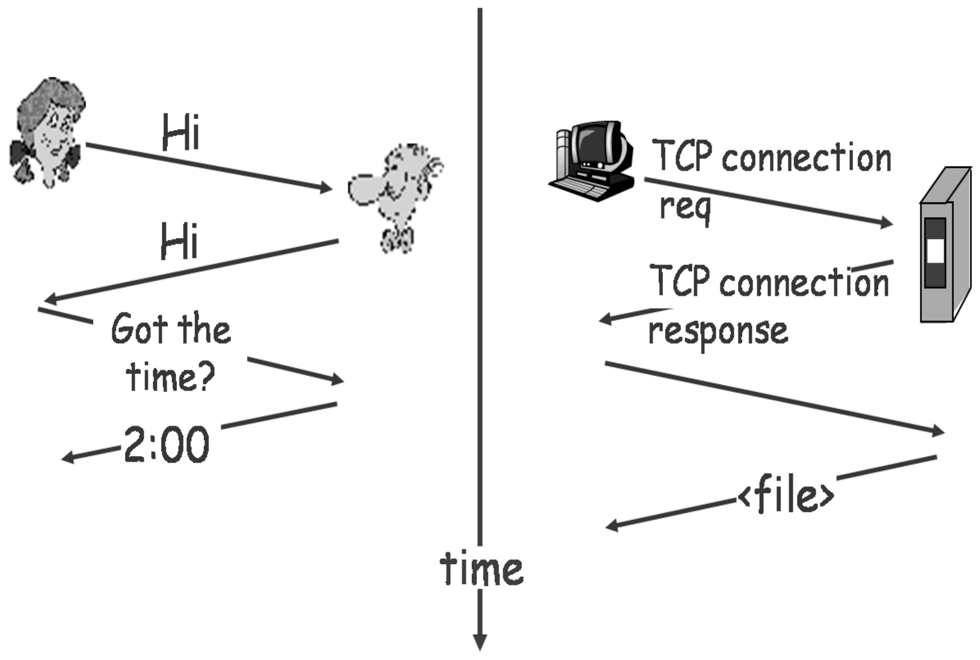


Figure 2-2 A human protocol and a computer network protocol

Note that in our human protocol, *there are specific messages we send, and specific actions we take in response to the received reply messages or other events (such as no reply within some given amount of time).* Clearly, transmitted and received messages, and actions taken when these messages are sent or received or other events occur, play a central role in a human protocol. If people run different protocols (for example, if one person has manners but the other does not, or if one understands the concept of time and the other does not) the protocols do not interoperate and no useful work can be accomplished. The same is true in networking it takes two (or more) communicating entities running the same protocol in order to accomplish a task.

2-3-2 Network Protocols

A network protocol is similar to a human protocol, except that the entities exchanging messages and taking actions are hardware or software components of some device (for example, computer, router, or other network-capable device). All activity in the Internet that involves two or more communicating remote entities is governed by a protocol. For example, protocols in routers determine a packet's path from source to destination; hardware-implemented protocols in the network interface cards of two physically connected computers control the flow of bits on the "wire" between the two network interface cards; congestion-control protocols in end systems control the rate at which packets are transmitted between sender and receiver. Protocols are running everywhere in the Internet.

As an example of a computer network protocol with which you are probably familiar, consider what happens when you make a request to a

Web server, that is, when you type in the URL of a Web page into your Web browser. The scenario is illustrated in the right half of Figure 2.2. First, your computer will send a "connection request" message to the Web server and wait for a reply. The Web server will eventually receive your connection request message and return a "connection reply" message. Knowing that it is now OK to request the Web document, your computer then sends the name of the Web page it wants to fetch from that Web server in a "GET" message. Finally, the Web server returns the Web page (file) to your computer.

A protocol *defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt, of a message or other event.*

2-3-3 Some Good Hyperlinks

• Internet Engineering Task Force (IETF), http://www.ietf.org: The IETF is an open international community concerned with the development and operation of the Internet and its architecture. The IETF was formally established by the Internet Architecture Board (IAB), http://www.iab.org, in 1986. The IETF meets three times a year; much of its ongoing work is conducted via mailing lists by working groups. The IETF is administered by the Internet Society, http://www.isoc.org, whose Web site contains lots of high-quality, Internet-related material.

• The World Wide Web Consortium (W3C), http://www.w3.org: The W3C was founded in 1994 to develop common protocols for the evolution of the World Wide Web. This is an outstanding site with fascinating information on emerging Web technologies, protocols, and standards.

• The Association for Computing Machinery (ACM), http://www.acm.org, and the Institute of Electrical and Electronics Engineers (IEEE), http://www.ieee.org: These are the two main international professional societies that have technical conferences, magazines, and journals in the networking area. The ACM Special Interest Group in Data Communications (SIGCOMM), http://www.acm.org/ sigcomm, the IEEE Communications Society, http://www.comsoc.org, and the IEEE Computer Society, http://www.computer.org, are the groups within these bodies whose efforts are most closely related to networking.

• Computer Networking: A Top-Down Approach Featuring the Internet (that is, the Web site for this textbook!), http://www.awl.com/kurose-ross: You'll find a wealth of resources at the Web site, including hyperlinks to relevant Web pages, Java applets illustrating networking concepts, homework problems with answers, programming projects, streaming audio lectures coupled to slides, and much more.

2-4 Network Edge

Edge network provides information exchange between the access network and the core network. The devices and facilities in the edge networks are switches, routers, routing switches, IADs and a variety of MAN/WAN devices, which are often called edge devices.

2.4.1 End Systems, Clients, and Servers

In computer networking jargon, the computers connected to the Internet are often referred to as **end systems.** They are referred to as end systems because they sit at the edge of the Internet, as shown in Figure 2.3. The Internet's end systems include many different types of computers. End users directly interface with some of these computers, including desktop computers (desktop PCs, Macs, and UNIX-based workstations) and mobile computers (portable computers and PDAs with wireless Internet connections). The Internet's end systems also include computers with which users do not directly interface, such as Web servers and e-mail servers. Furthermore, an increasing number of alternative devices, such as thin clients and household appliances, Web TVs and set top boxes, and digital cameras are being attached to the Internet as end systems.

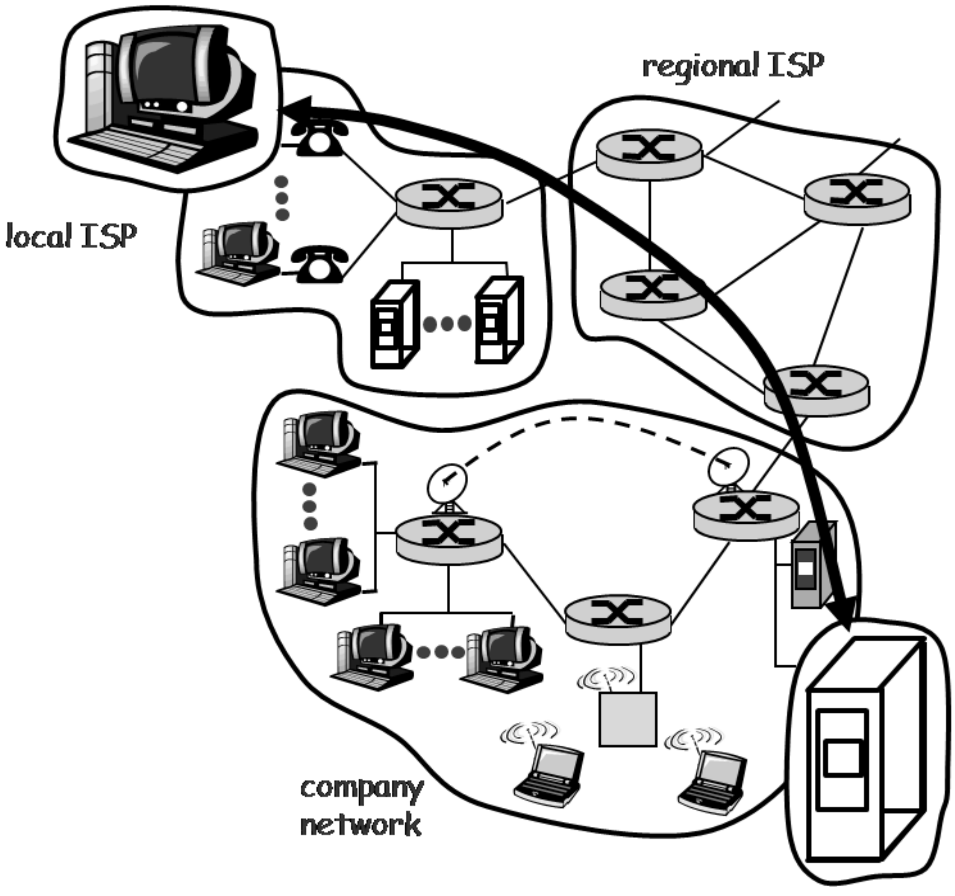


Figure 2-3 End-system interaction

End systems are also referred to as **hosts** because they host (that is, run) application programs such as a Web browser program, a Web server program, an e-mail reader program, or an e-mail server program. Hosts are sometimes further divided into two categories: **clients** and **servers.**

A **client program** is a program running on one end system that requests and receives a service from a **server program** running on another end system. A client program typically runs on one computer and the server program runs on another computer, client/server Internet applications are, by definition, **distributed applications.** The client program and the server program interact with each other by sending each other messages over the Internet. At this level of abstraction, the routers, links and other "nuts and bolts" of the Internet serve as a "black box" that transfers messages between the distributed, communicating components of an Internet application. This is the level of abstraction depicted in Figure 2.3.

2.4.2 Connectionless and Connection-Oriented Service

TCP/IP networks, and in particular the Internet, provide two types of services to end-system applications: **connectionless service** and **connection-oriented service.**

When an application uses the connection-oriented service, the client program and the server program (residing in different end systems) send control packets to each other before sending packets with real data (such as e-mail messages). This so called handshaking procedure alerts the client and server, allowing them to prepare for an onslaught of packets. Once the handshaking procedure is finished, a connection is said to be established between the two end systems. The Internet's connection-oriented service comes bundled with several other services, including reliable data transfer, flow control, and congestion control. The Internet's connection-oriented service has a name TCP

There is no handshaking with the Internet's connectionless service. When one side of an application wants to send packets to the other side of the application, the sending program simply sends the packets, since there is no handshaking procedure prior to data packet transmission, data can be delivered sooner. But there is no reliable data transfer either, so a source never knows for sure which packets have arrived at the destination. *The* Internet's connectionless service is called UDP.

2-5 Network Core

The network core meshes of routers that interconnect the Internet's end systems. Figure 2.4 highlights the network core with thick, shaded lines.

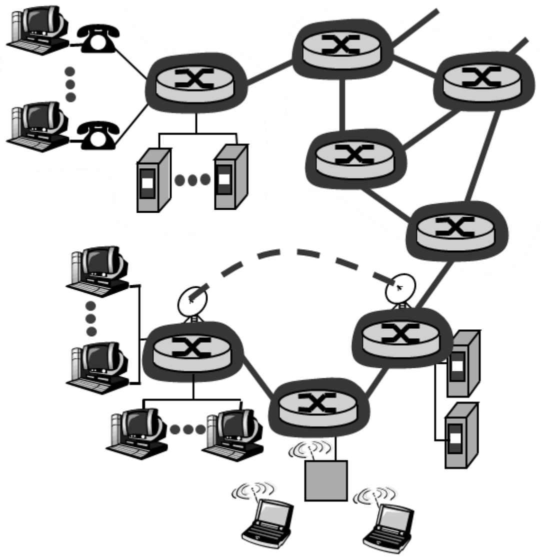


Figure 2-4 network core

2-5-1 Circuit Switching and Packet Switching

A core network is a central network into which other networks feed. There are two fundamental approaches toward building a network core: **circuit switching** and **packet switching.**

In **circuit switching,** the method used for placing telephone calls, the signal is routed through various nodes from the point of origin to the point of destination, and the selected path is fixed during the duration of the session. The path is also dedicated once a session (phone call) is initiated then there can be no other users (callers) on the channel connecting the sender and receiver.

In **packet switching** information to be sent is divided into small blocks called packets. Each packet is coded with the destination address and is sent separately over the network. Different packets can take different routes through the network to reach the destination node. As each node receives a packet, it checks the destination address on the packet and either accepts the packet or sends it further along if it is intended for another recipient. Upon arrival at the destination, the packets are assembled together in the correct order and presented at the receiver's terminal. Unlike circuit switching, packet switching does not tie up a channel since there is no dedicated path from the sender to the receiver. Packet switching thus results in more efficient use of a network than circuit switching. The method of transferring data over the Internet, the foremost "public" wide area network, is packet switching. When you send an email message over the Internet, the message is formed into packets and sent from node to node until it reaches the final destination.

2-5-2 Packet Forwarding in Computer Networks

There are two broad classes of packet-switched networks: datagram networks and virtual circuit networks. They differ in whether their switches use destination addresses or so-called virtual circuit numbers to forward packets toward their destinations. We'll call any network that forwards packets according to host destination addresses a **datagram network.** The routers in the Internet forward packets according to host destination addresses; hence the Internet is a datagram network. We'll call any network that forwards packets according to virtual circuit numbers a **virtual circuit network.** Examples of packet-switching technologies that use virtual circuits include X.25, frame relay, and ATM (asynchronous transfer mode).

2-6 Network Access and Physical Media

Network access is that part of a communications network which connects subscribers to their immediate service provider. It is contrasted with the *core network.* Network Access Network access can be loosely classified into three categories:

• **Residential access,** connecting home end systems into the network

• **Company access,** connecting end systems in a business or educational institution into the network

• **Mobile access,** connecting mobile end systems into the network

Data is transmitted over copper wires, fiber optic cable, radio and microwaves. The term 'media' is used to generically refer to the physical connectors, wires or devices used to plug things together. The physical medium can take many shapes and forms and does not have to be of the same type for each transmitter-receiver pair along the path. Examples of physical media include twisted-pair copper wire, coaxial cable, multi-mode fiber-optic cable, terrestrial radio spectrum, and satellite radio spectrum. Physical media fall into two categories: **guided media** and **unguided media.** With guided media, the waves are guided along a solid medium, such as a fiber-optic cable, a twisted-pair copper wire, or a coaxial cable. With unguided media, the waves propagate in the atmosphere and in outer space, such as in a wireless LAN or a digital satellite channel.

2-7 Delay and Loss in Packet-Switched Networks

A packet travels from one node (host or router) to the subsequent node (host or router) along this path; the packet suffers from several different types of delays at *each* node along the path. The most important of these delays are the **nodal processing delay, queuing delay, transmission delay,** and **propagation delay;** together, these delays accumulate to give a **total nodal delay.**

Let us explore these delays in the context of Figure 2-5. As part of its end-to-end route between source and destination, a packet is sent from the upstream node through router A to router B. Our goal is to characterize the nodal delay at router A. Note that router A has an outbound link leading to router B. This link is preceded by a queue (also known as a buffer). When the packet arrives at router A from the up-stream node, router A examines the packet's header to determine the appropriate outbound link for the packet, and then directs the packet to the link. In this example, the outbound link for the packet is the one that leads to router B. A packet can be transmitted on a link only if there is no other packet currently being transmitted on the link and if there are no other packets preceding it in the queue; if the link is currently busy or if there are other packets already queued for the link, the newly arriving packet will then join the queue.

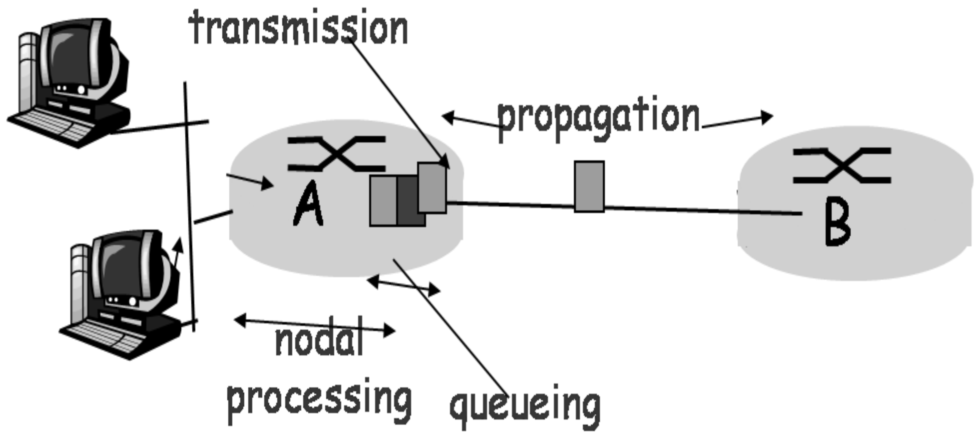


Figure 2-5 The nodal delay at router

3. Layered Network Architecture

Structure

Objectives

- State the requirement for layered approach
- Explain the basic concept of layering in the network model
- Define entities protocols in networking context
- Describe ISO's OSI Reference Model
- Explain information flow in OSI references Model.
- Explain functions of the seven layers of OSI Model

3.1 Basic concept of layering

Network architectures define the standards and techniques for designing and building communication systems for computers and other devices. In the past, vendors developed their own architectures and required that other vendors conform to this architecture if they wanted to develop compatible hardware and software. There are proprietary network architectures such as IBM's SNA (Systems Network Architecture) and there are open architectures like the OSI (Open Systems Interconnection) model defined by the International Organization for Standardization. The previous strategy, where the computer network is designed with the hardware as the main concern and software is afterthought, no longer works. Network software is now highly *structured.*

To reduce the design complexity, most of the networks are organized as a series of **layers** or **levels**, each one build upon one below it. The basic idea of a layered architecture is *to* divide the design into small pieces. Each layer adds to the services provided by the lower layers in such a manner that the highest layer is provided a full set of services to manage communications and run the applications. The benefits of the layered models are modularity and clear interfaces, i.e. open architecture and comparability between the different providers' components.

A basic principle is to ensure independence of layers by defining services provided by each layer to the next higher layer without defining how the services are to be performed. This permits changes in a layer without affecting other layers. Prior to the use of layered protocol architectures, simple changes such as adding one terminal type to the list of those supported by architecture often required changes to essentially all communications software at a site. The number of layers, functions and contents of each layer differ from network to network. However in all networks, the purpose of each layer is to offer certain services to higher layers, shielding those layers from the details of how the services are actually implemented.

The basic elements of a layered model are services, protocols and interfaces. A *service* is a set of actions that a layer offers to another (higher) layer. *Protocol* is a set of rules that a layer uses to exchange information with a peer entity. These rules concern both the contents and the order of the messages used. Between the layers service interfaces are defined. The messages from one layer to another are sent through those interfaces.

In n-layer architecture, layer n on one machine carries on conversation with the layer n. on other machine. The rules and conventions used in this conversation are collectively known as the layer-*n protocol*. Basically, a protocol is an agreement between the communicating parties on how communication is to proceed. Violating the protocol will make communication more difficult, if not impossible. Five-layer architecture is shown in Fig.3.1**;** the entities comprising the corresponding layers on different machines are called *peers*. In other words, it is the peers that communicate using protocols. In reality, no data is transferred from layer n on one machine to layer n of another machine. Instead, each layer passes data and control information to the layer immediately below it, until the lowest layer is reached. Below layer-1 is the physical layer through which actual communication occurs. The peer process abstraction is crucial to all network design. Using it, the un-manageable tasks of designing the complete network can be broken into several smaller, manageable, design problems, namely design of individual layers.

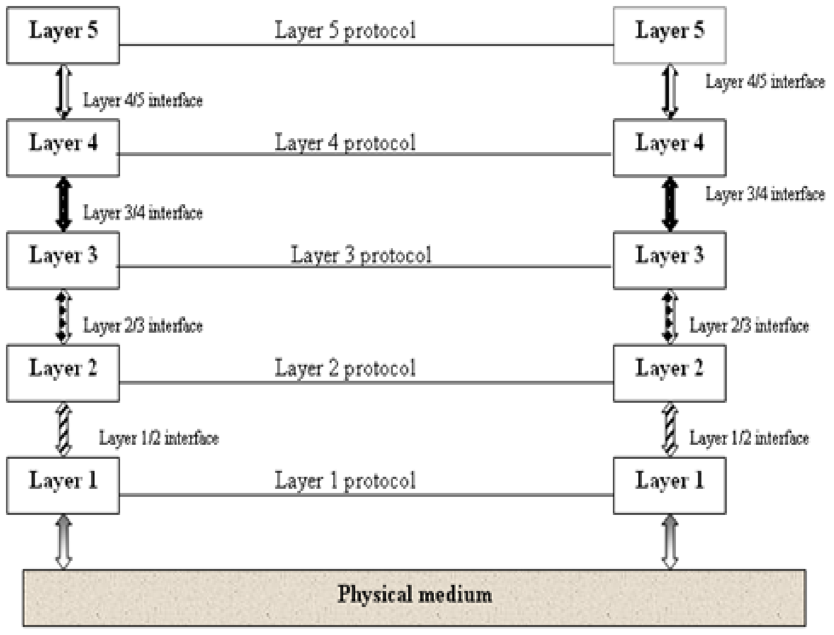


Figure 3.1 Basic five layer architecture

Between each pair of adjacent layers there is an **interface**. The *interface* defines which primitives operations and services the lower layer offers to the upper layer adjacent to it. When network designer decides how many layers to include in the network and what each layer should do, one of the main considerations is defining clean interfaces between adjacent layers. Doing so, in turns requires that each layer should perform well-defined functions. In addition to minimize the amount of information passed between layers, clean-cut interface also makes it simpler to replace the implementation of one layer with a completely different implementation, because all what is required of new implementation is that it offers same set of services to its upstairs neighbor as the old implementation (that is what a layer provides and how to use that service from it is more important than knowing how exactly it implements it).

A set of layers and protocols is known as **network architecture**. The specification of architecture must contain enough information to allow an implementation to write the program or build the hardware for each layer so that it will correctly obey the appropriate protocol. Neither the details of implementation nor the specification of interface is a part of network architecture because these are hidden away inside machines and not visible from outside. It is not even necessary that the interface on all machines in a network be same, provided that each machine can correctly use all protocols. A list of protocols used by a certain system, one protocol per layer, is called **protocol stack**.

Why Layered architecture?

1. To make the design process easy by breaking unmanageable tasks into several smaller and manageable tasks (by divide-and-conquer approach).

2. Modularity and clear interfaces, so as to provide comparability between the different providers' components.

3. Ensure independence of layers, so that implementation of each layer can be changed or modified without affecting other layers.

4. Each layer can be analyzed and tested independently of all other layers.

3.2 Open System Interconnection Reference Model

The Open System Interconnection (OSI) reference model describes how information from a software application in one computer moves through a network medium to a software application in another computer. The OSI reference model is a conceptual model composed of seven layers, each specifying particular network functions. The model was developed by the International Organization for Standardization (ISO) in 1984, and it is now considered the primary architectural model for inter computer communications. The OSI model divides the tasks involved with moving information between networked computers into seven smaller, more manageable task groups. A task or group of tasks is then assigned to each of the seven OSI layers. Each layer is reasonably self-contained so that the tasks assigned to each layer can be implemented independently. This enables the solutions offered by one layer to be updated without adversely affecting the other layers.

The OSI Reference Model includes seven layers:

1. ***Application Layer***: Provides Applications with access to network services.

2. ***Presentation Layer***: Determines the format used to exchange data among networked computers.

3. ***Session Layer***: Allows two applications to establish, use and disconnect a connection between them called a session. Provides name recognition and additional functions like security, which are needed to allow applications to communicate over the network.

4. ***Transport Layer***: Ensures that data is delivered error free, in sequence and with no loss, duplications or corruption. This layer also repackages data by assembling long messages into lots of smaller messages for sending, and repackaging the smaller messages into the original larger message at the receiving end.

5. ***Network Layer***: This is responsible for addressing messages and data so they are sent to the correct destination, and for translating logical addresses and names (like a machine name FLAME) into physical addresses. This layer is also responsible for finding a path through the network to the destination computer.

6. ***Data-Link Layer***: This layer takes the data frames or messages from the Network Layer and provides for their actual transmission. At the receiving computer, this layer receives the incoming data and sends it to the network layer for handling. The Data-Link Layer also provides error-free delivery of data between the two computers by using the physical layer. It does this by packaging the data from the Network Layer into a frame, which includes error detection information. At the receiving computer, the Data-Link Layer reads the incoming frame, and generates its own error detection information based on the received frames data. After receiving the entire frame, it then compares its error detection value with that of the incoming frames, and if they match, the frame has been received correctly.

7. ***Physical Layer***: Controls the transmission of the actual data onto the network cable. It defines the electrical signals, line states and

encoding of the data and the connector types used. An example is 10BaseT.

3.2.1 Characteristics of the OSI Layers

The seven layers of the OSI reference model can be divided into two categories: upper layers and lower layers as shown in Fig. 3.2.

The upper layers of the OSI model deal with application issues and generally are implemented only in software. The highest layer, the application layer, is closest to the end user. Both users and application layer processes interact with software applications that contain a communications component. The term upper layer is sometimes used to refer to any layer above another layer in the OSI model.

The lower layers of the OSI model handle data transport issues. The physical layer and the data link layer are implemented in hardware and software. The lowest layer, the physical layer, is closest to the physical network medium (the network cabling, for example) and is responsible for actually placing information on the medium.

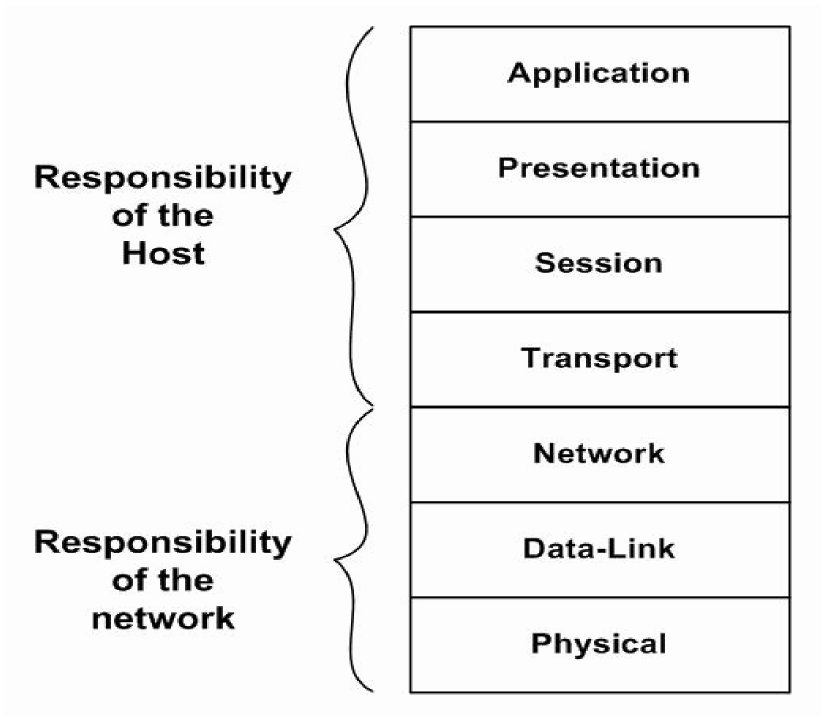


Figure 3.2 Two sets of layers make up the OSI layers

3.2.2 Protocols

The OSI model provides a conceptual framework for communication between computers, but the model itself is not a method of communication. Actual communication is made possible by using communication protocols. In the context of data networking, a **protocol** *is a formal set of rules and conventions that governs how computers exchange information over a network medium*. A protocol implements the functions of one or more of the OSI layers.

A wide variety of communication protocols exist. Some of these protocols include LAN protocols, WAN protocols, network protocols, and routing protocols. LAN protocols operate at the physical and data link layers of the OSI model and define communication over various LAN media. WAN protocols operate at the lowest three layers of the OSI model and define communication over the various wide-area media. Routing protocols are network layer protocols that are responsible for exchanging information between routers so that the routers can select the proper path for network traffic. Finally, network protocols are the various upper-layer protocols that exist in a given protocol suite. Many protocols rely on others for operation. For example, many routing protocols use network protocols to exchange information between routers. This concept of building upon the layers already in existence is the foundation of the OSI model.

3.2.3 OSI Model and Communication between Systems

Information being transferred from a software application in one computer system to a software application in another must pass through the OSI layers. For example, if a software application in System A has information to transmit to a software application in System B, the application program in System A will pass its information to the application layer (Layer 7) of System A. The application layer then passes the information to the presentation layer (Layer 6), which relays the data to the session layer (Layer 5), and so on down to the physical layer (Layer 1). At the physical layer, the information is placed on the physical network medium and is sent across the medium to System B. The physical layer of System B removes the information from the physical medium, and then its physical layer passes the information up to the data link layer (Layer 2), which passes it to the network layer (Layer 3), and so on, until it reaches the application layer (Layer 7) of System B. Finally, the application layer of System B passes the information to the recipient application program to complete the communication process.

3.2.4 Interaction between OSI Model Layers

A given layer in the OSI model generally communicates with three other OSI layers: the layer directly above it, the layer directly below it, and its peer layer in other networked computer systems. The data link layer in System A, for example, communicates with the network layer of System A, the physical layer of System A, and the data link layer in System B. Figure3.3 illustrates this example.

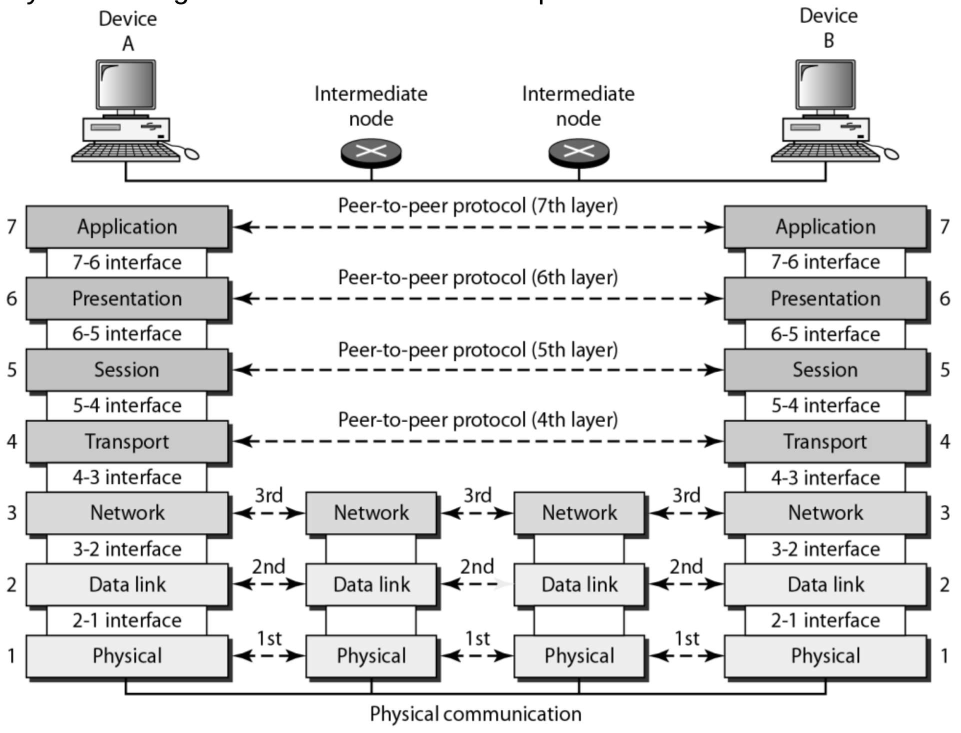


Figure 3.3 OSI Model Layers Communicate with Other Layers

3.3 Services and service access points

One OSI layer communicates with another layer to make use of the services provided by the second layer. The services provided by adjacent layers help a given OSI layer communicate with its peer layer in other computer systems. Three basic elements are involved in layer services: the service user, the service provider, and the service access point (SAP).

In this context, the service user is the OSI layer that requests services from an adjacent OSI layer. The service provider is the OSI layer that provides services to service users. OSI layers can provide services to multiple service users. The SAP is a conceptual location at which one OSI layer can request the services of another OSI layer.

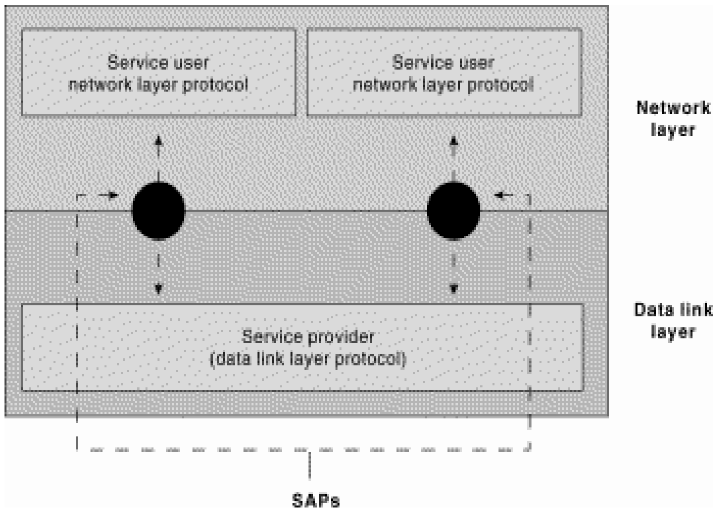

Service user network layer protocol
Service user network layer protocol
Network layer
Data link layer
Service provider (data link layer protocol)
SAPs

Figure 3.4 Service Users, Providers, and SAPs interact at the Network and Data Link Layers

3.3.1 OSI Model Layers and Information Exchange

The seven OSI layers use various forms of control information to communicate with their peer layers in other computer systems. This control information consists of specific requests and instructions that are exchanged between peer OSI layers. Control information typically takes one of two forms: headers and trailers. Headers are pre-ended to data that has been passed down from upper layers. Trailers are appended to data that has been passed down from upper layers. An OSI layer is not required to attach a header or a trailer to data from upper layers. Headers, trailers, and data are relative concepts, depending on the layer that analyzes the information unit. At the network layer, for example, an information unit consists of a Layer 3 header and data. At the data link layer, however, all the information passed down by the network layer (the Layer 3 header and the data) is treated as data.

In other words, the data portion of an information unit at a given OSI layer potentially can contain headers, trailers, and data from all the higher layers. This is known as encapsulation. Figure 3-5 shows how the header and data from one layer are encapsulated into the header of the next lowest layer.

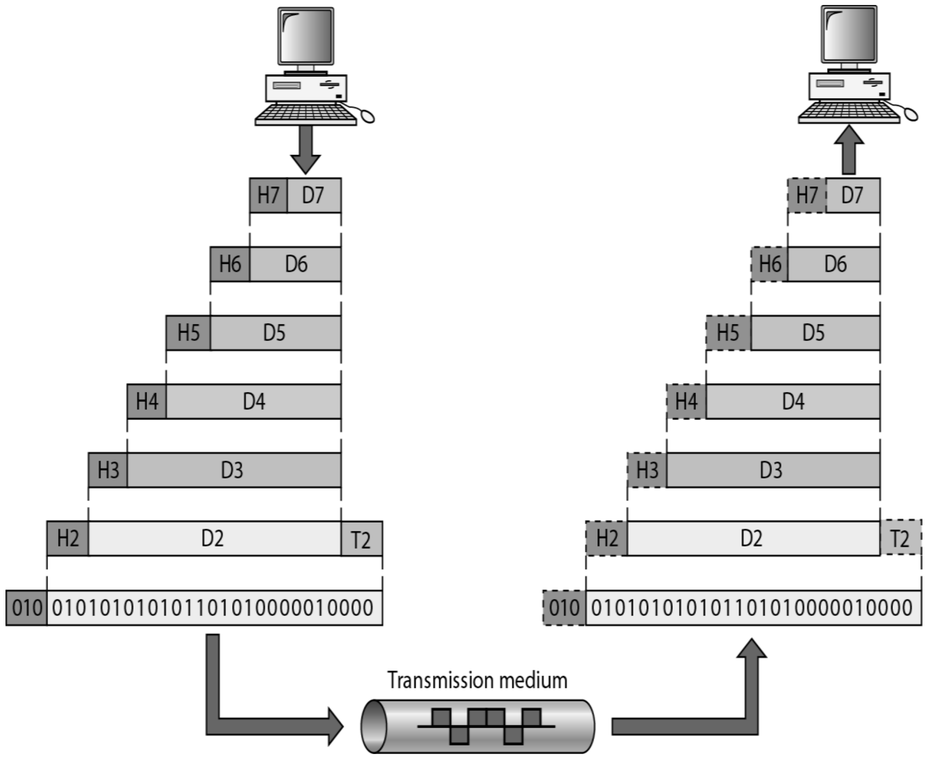


Figure 3.5 Headers and Data can be encapsulated during Information exchange

3.3.2 Information Exchange Process

The information exchange process occurs between peer OSI layers. Each layer in the source system adds control information to data, and each layer in the destination system analyzes and removes the control information from that data. If system A has data from software application to send to System B, the data is passed to the application layer. The application layer in System A then communicates any control information required by the application layer in System B by pre-pending a header to the data. The resulting information unit (a header and the data) is passed to the presentation layer, which pre-ends its own header containing control information intended for the presentation layer in System B. The information unit grows in size as each layer pre-ends its own header (and, in some cases, a trailer) that contains control information to be used by its peer layer in System B. At the physical layer, the entire information unit is placed onto the network medium.

The physical layer in System B receives the information unit and passes it to the data link layer. The data link layer in System B then reads the control information contained in the header pre-pended by the data link layer in System A. The header is then removed, and the remainder of the information unit is passed to the network layer. Each layer performs the same actions: The layer reads the header from its peer layer, strips it off, and passes the remaining information unit to the next highest layer. After the application layer performs these actions, the data is passed to the recipient software application in System B, in exactly the form in which it was transmitted by the application in System A.

3.4 Functions of the OSI Layers

Functions of different layers of the OSI model are presented in this section.

3.4.1 Physical Layer

The physical layer is concerned with transmission of raw bits over a communication channel. It specifies the mechanical, electrical and procedural network interface specifications and the physical transmission of bit streams over a transmission medium connecting two pieces of communication equipment. In simple terns, the physical layer decides the following:

- Number of pins and functions of each pin of the network connector (Mechanical)
- Signal Level, Data rate (Electrical)
- Whether simultaneous transmission in both directions
- Establishing and breaking of connection
- Deals with physical transmission

There exist a variety of physical layer protocols such as RS-232C, Rs-449 standards developed by Electronics Industries Association (EIA). Figure 2-6 shows the position of the physical layer with respect to the transmission medium and the data link layer.

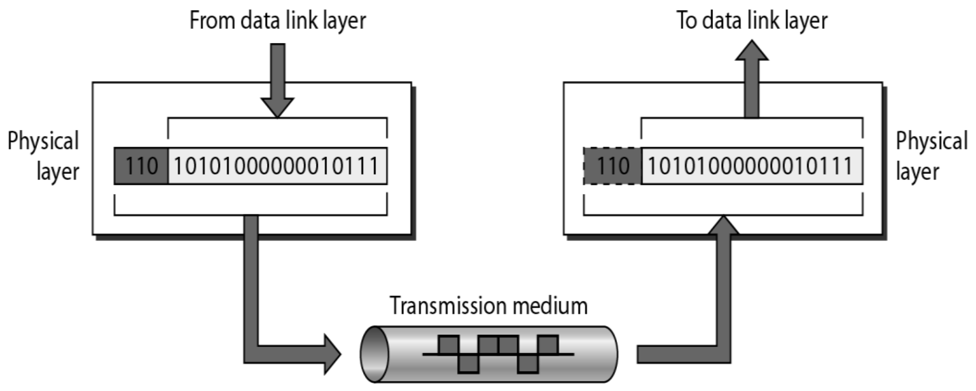


Figure 3.6 *physical layer*

3.4.2 Data Link Layer

The goal of the data link layer is to provide reliable, efficient communication between adjacent machines connected by a single communication channel. Specifically:

1. Group the physical layer bit stream into units called frames. Note that frames are nothing more than ``packets'' or ``messages''. By convention, we shall use the term ``frames'' when discussing DLL packets.

2. Sender calculates the checksum and sends checksum together with data. The checksum allows the receiver to determine when a frame has been damaged in transit or received correctly.

3. Receiver re-computes the checksum and compares it with the received value. If they differ, an error has occurred and the frame is discarded.

4. Error control protocol returns a positive or negative acknowledgment to the sender. A positive acknowledgment indicates the frame was received without errors, while a negative acknowledgment indicates the opposite.

5. Flow control prevents a fast sender from overwhelming a slower receiver. For example, a supercomputer can easily generate data faster than a PC can consume it.

6. In general, data link layer provides service to the network layer. The network layer wants to be able to send packets to its neighbors without worrying about the details of getting it there in one piece.

Figure 3-7 shows the relationship of the data link layer to the network and physical layer

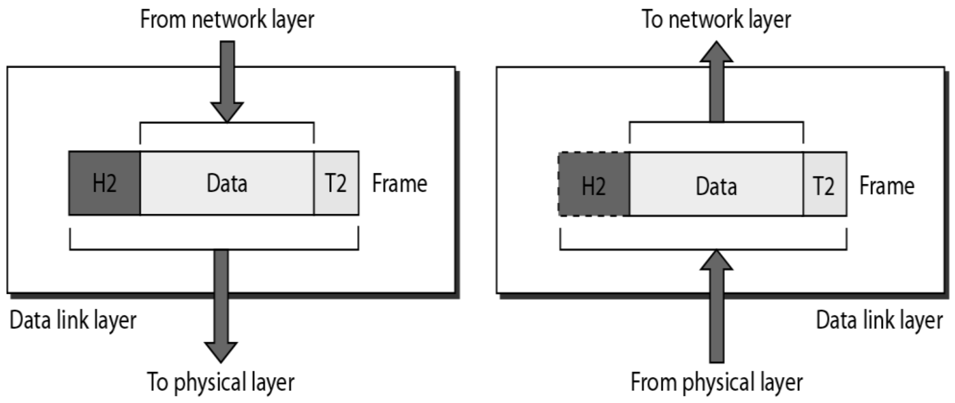


Figure 3.7 Data link layer

3.4.2.1 Design Issues

Below are the some of the important design issues of the data link layer:

a) Reliable Delivery:

Frames are delivered to the receiver reliably and in the same order as generated by the sender. Connection state keeps track of sending order and which frames require retransmission. For example, receiver state includes which frames have been received, which ones have not, etc.

b) Best Effort:

The receiver does not return acknowledgments to the sender, so the sender has no way of knowing if a frame has been successfully delivered.

When would such a service be appropriate?

1. When higher layers can recover from errors with little loss in performance. That is, when errors are so infrequent that there is little to be gained by the data link layer performing the recovery. It is just as easy to have higher layers deal with occasional loss of packet.

2. For real-time applications requiring ``better never than late" semantics. Old data may be worse than no data.

c) Acknowledged Delivery

The receiver returns an acknowledgment frame to the sender indicating that a data frame was properly received. This sits somewhere between the other two in that the sender keeps connection state, but may not necessarily retransmit unacknowledged frames. Likewise, the receiver may hand over received packets to higher layer in the order in which they arrive, regardless of the original sending order. Typically, each frame is assigned a unique sequence number, which the receiver returns in an acknowledgment frame to indicate which frame the ACK refers to the sender must retransmit unacknowledged (e.g., lost or damaged) frames.

d) Framing

The DLL translates the physical layer's raw bit stream into discrete units (messages) called *frames*. How can the receiver detect frame boundaries? Various techniques are used for this: Length Count, Bit Stuffing, and Character stuffing.

e) Error Control

Error control is concerned with insuring that all frames are eventually delivered (possibly in order) to a destination. To achieve this, three items are required: Acknowledgements, Timers, and Sequence Numbers.

f) Flow Control

Flow control deals with throttling the speed of the sender to match that of the receiver. Usually, this is a dynamic process, as the receiving speed depends on such changing factors as the load, and availability of buffer space.

3.4.2.2 Link Management

In some cases, the data link layer service must be ``opened" before use:

- The data link layer uses open operations for allocating buffer space, control blocks, agreeing on the maximum message size, etc.
- Synchronize and initialize send and receive sequence numbers with its peer at the other end of the communications channel.

3.4.2.3 Error Detection and Correction

In data communication, error may occur because of various reasons including attenuation, noise. Moreover, error usually occurs as bursts rather than independent, single bit errors. For example, a burst of lightning will affect a set of bits for a short time after the lightning strike. Detecting and correcting errors requires redundancy (i.e., sending additional information along with the data).

There are two types of attacks against errors:

• Error Detecting Codes: Include enough redundancy bits to detect errors and use ACKs and retransmissions recover from the errors. Example: parity encoding.
• Error Correcting Codes: Include enough redundancy to detect and correct errors. Examples: CRC checksum, MD5.

3.4.3 Network Layer

The basic purpose of the network layer is to provide an end-to-end communication capability in contrast to machine-to-machine communication provided by the data link layer. This end-to-end is performed using two basic approaches known as connection- oriented or connectionless network-layer services. Figure 3-8 shows the relationship of the network layer to the data link and transport layer.

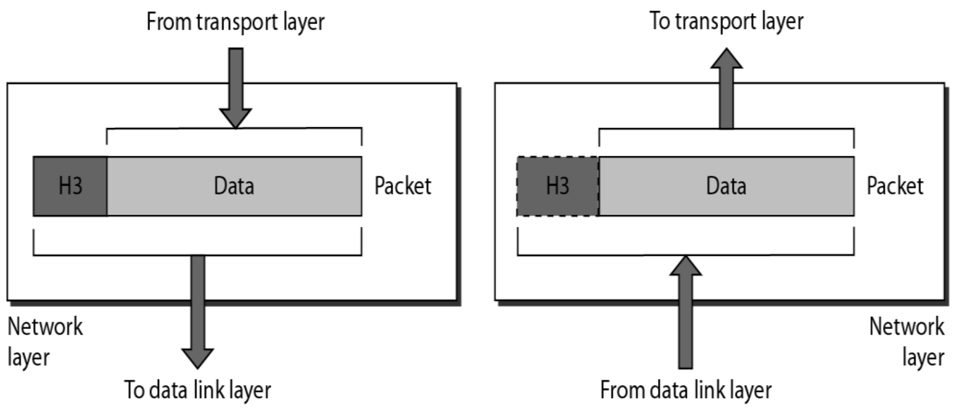


Figure 3.8 *Network layer*

3.4.3.1 Four issues:

1. Interface between the host and the network (the network layer is typically the boundary between the host and subnet)

2. Routing

3. Congestion and deadlock

4. Internetworking (A path may traverse different network technologies (e.g., Ethernet, point-to-point links, etc.)

3.4.3.2 Network Layer Interface

There are two basic approaches used for sending packets, which is a group of bits that includes data plus source and destination addresses, from node to node called *virtual circuit* and *datagram* methods. These are also referred to as *connection-oriented* and *connectionless* network-layer services. In virtual circuit approach, a *route*, which consists of logical connection, is first established between two users. During this establishment phase, the two users not only agree to set up a connection between them but also decide upon the quality of service to be associated with the connection. The well-known virtual- circuit protocol is the ISO and CCITT *X.25* specification. The datagram is a self- contained message unit, which contains sufficient information for routing from the source node to the destination node without dependence on previous message interchanges between them. In contrast to the virtual-circuit method, where a fixed path is explicitly set up before message transmission, sequentially transmitted messages can follow completely different paths. The datagram method is analogous to the postal system and the virtual-circuit method is analogous to the telephone system.

3.4.3.3 Overview of Other Network Layer Issues:

The network layer is responsible for routing packets from the source to destination. The *routing algorithm* is the piece of software that decides where a packet goes next (e.g., which output line, or which node on a broadcast channel).

For connectionless networks, the routing decision is made for each datagram. For connection-oriented networks, the decision is made once, at circuit setup time.

3.4.3.4 Routing Issues:

The routing algorithm must deal with the following issues:

- Correctness and simplicity: networks are never taken down; individual parts (e.g., links, routers) may fail, but the whole network should not.
- Stability: if a link or router fails, how much time elapses before the remaining routers recognize the topology change? (Some never do.)
- Fairness and optimality: an inherently intractable problem. Definition of optimality usually doesn't consider fairness. Do we want to maximize channel usage? Minimize average delay?

When we look at routing in detail, we'll consider both adaptive--those that take current traffic and topology into consideration--and non-adaptive algorithms.

3.4.3.4 Congestion

The network layer also must deal with congestion:

- When more packets enter an area than can be processed, delays increase and performance decreases. If the situation continues, the subnet may have no alternative but to discard packets.
- If the delay increases, the sender may (incorrectly) retransmit, making a bad situation even worse.
- Overall, performance degrades because the network is using (wasting) resources processing packets that eventually get discarded.

3.4.3.5 Internetworking

Finally, when we consider internetworking -- connecting different network technologies together -- one finds the same problems, only worse:

- Packets may travel through many different networks
- Each network may have a different frame format
- Some networks may be connectionless, other connection oriented

3.4.3.6 Routing

Routing is concerned with the question: Which line should router J use when forwarding a packet to router K.

There are two types of algorithms:

- **Adaptive algorithms** use such dynamic information as current topology, load, delay, etc. to select routes.
- In **non-adaptive algorithms**, routes never change once initial routes have been selected called static routing.

Obviously, adaptive algorithms are more interesting, as non-adaptive algorithms don't even make an attempt to handle failed links.

3.4.4 Transport Layer

The transport level provides end-to-end communication between processes executing on different machines. Although the services provided by a transport protocol are similar to those provided by a data link layer protocol, there are several important differences between the transport and lower layers:

***1. User Oriented*.** Application programmers interact directly with the transport layer, and from the programmers perspective, the transport layer is the ``network''. Thus the transport layer should be oriented more towards user services than simply reflect what the underlying layers happen to provide.

2. Negotiation of Quality and Type of Services. The user and transport protocol may need to negotiate as to the quality or type of service to be provided. Examples A user may want to negotiate such options as: throughput, delay, protection, priority, reliability, etc.

3. Guarantee Service. The transport layer may have to overcome service deficiencies of the lower layers (e.g. providing reliable service over an unreliable network layer).

4. Addressing becomes a significant issue. That is, now the user must deal with it; before it was buried in lower levels.

Two solutions:

- Use well-known addresses that rarely if ever change, allowing programs to ``wire in'' addresses. For what types of service does this work? While this works for services that are well established (e.g., mail, or telnet), it doesn't allow a user to easily experiment with new services.

- Use a name server. Servers register services with the name server, which clients contact to find the transport address of a given service.

In both cases, we need a mechanism for mapping high-level service names into low-level encoding that can be used within packet headers of the network protocols. In its general form, the problem is quite complex. One simplification is to break the problem into two parts: have transport addresses be a combination of machine address and local process on that machine.

***5. Storage capacity* of the subnet.** Assumptions valid at the data link layer do not necessarily hold at the transport Layer. Specifically, the subnet may buffer messages for a potentially long time, and an ``old" packet may arrive at a destination at unexpected times.

6. We need a dynamic flow control mechanism. The data link layer solution of reallocating buffers is inappropriate because a machine may have hundreds of connections sharing a single physical link. In addition, appropriate settings for the flow control parameters depend on the communicating end points (e.g., Cray supercomputers vs. PCs), not on the protocol used.

Don't send data unless there is room. Also, the network layer/data link layer solution of simply not acknowledging frames for which the receiver has no space is unacceptable. Why? In the data link case, the line is not being used for anything else; thus retransmissions are inexpensive. At the transport level, end-to-end retransmissions are needed, which wastes resources by sending the same packet over the same links multiple times. If the receiver has no buffer space, the sender should be prevented from sending data.

7. *Deal with congestion control*. In connectionless Internets, transport protocols must exercise congestion control. When the network becomes congested, they must reduce rate at which they insert packets into the subnet, because the subnet has no way to prevent itself from becoming overloaded.

8. *Connection establishment*. Transport level protocols go through three phases: establishing, using, and terminating a connection. For data gram-oriented protocols, opening a connection simply allocates and initializes data structures in the operating system kernel.

Connection oriented protocols often exchanges messages that negotiate options with the remote peer at the time a connection are opened. Establishing a connection may be tricky because of the possibility of old or duplicate packets.

Finally, although not as difficult as establishing a connection, terminating a connection presents subtleties too. For instance, both ends of the connection must be sure that all the data in their queues have been delivered to the remote application. Figure 3-9 shows the relationship of the transport layer to the network and session layer.

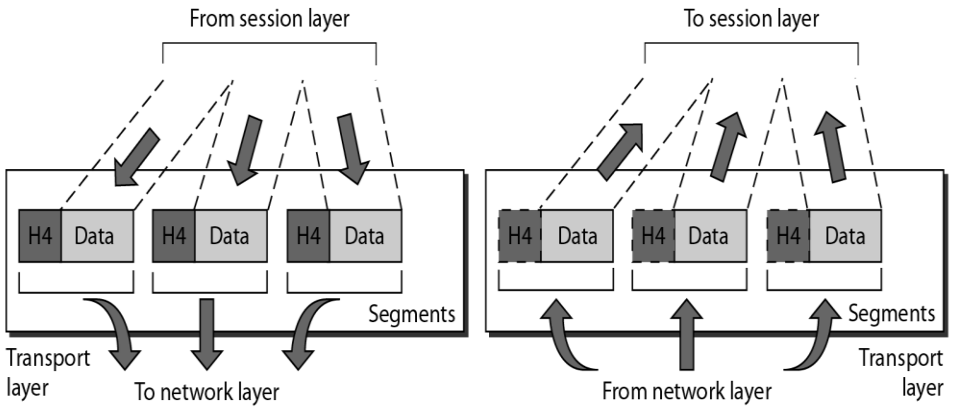


Figure 3.9 *Transport layer*

3.4.5 Session Layer

This layer allows users on different machines to establish session between them. A session allows ordinary data transport but it also provides enhanced services useful in some applications. A session may be used to allow a user to log into a remote time sharing machine or to transfer a file between two machines. Some of the session related services are:

1. *Dialogue Control*. Session can allow traffic to go in both direction at the same time, or in only one direction at one time.

2. *Token management*. For some protocols, it is required that both sides don't attempt same operation at the same time. To manage these activities, the session layer provides tokens that can be exchanged. Only one side that is holding token can perform the critical operation. This concept can be seen as entering into a critical section in operating system using semaphores.

3. *Synchronization*. Consider the problem that might occur when trying to transfer a 4- hour file transfer with a 2-hour mean time between crashes. After each transfer was aborted, the whole transfer has to start again and again would probably fail. To eliminate this problem, Session layer provides a way to insert checkpoints into data streams, so that after a crash, only the data transferred after the last checkpoint have to be repeated. Figure 3-10 shows the relationship of the session layer to the transport and presentation layer.

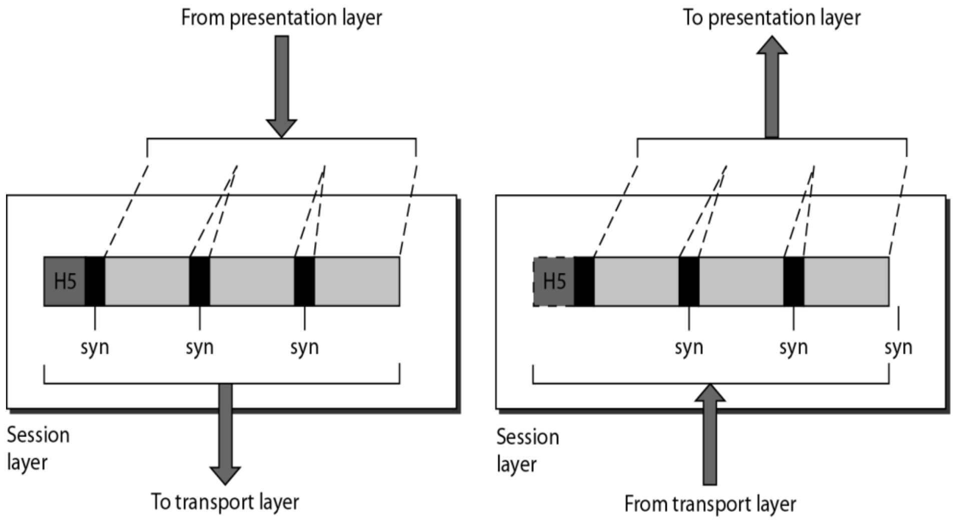


Figure 3.10 *Session layer*

3.4.6 Presentation Layer

This layer is concerned with Syntax and Semantics of the information transmitted, unlike other layers, which are interested in moving data reliably from one machine to other. Few of the services that Presentation layer provides are:

1. Encoding data in a standard agreed upon way.
2. It manages the abstract data structures and converts from representation used inside computer to network standard representation and back.

Figure 2-11 shows the relationship of the presentation layer to the session and application layer.

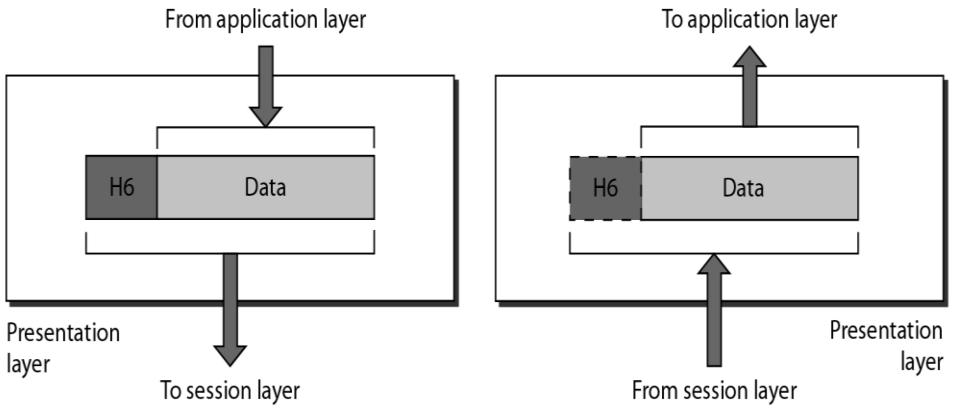


Figure 3.11 *Presentation layer*

3.4.7 Application Layer

The application layer consists of what most users think of as programs. The application does the actual work at hand. Although each application is different, some applications are so useful that they have become standardized. The Internet has defined standards for:

- File transfer (FTP): Connect to a remote machine and send or fetch an arbitrary file. FTP deals with authentication, listing a directory contents, ASCII or binary files, etc.
- Remote login (telnet): A remote terminal protocol that allows a user at one site to establish a TCP connection to another site, and then pass keystrokes from the local host to the remote host.
- Mail (SMTP): Allow a mail delivery agent on a local machine to connect to a mail delivery agent on a remote machine and deliver mail.
- News (NNTP): Allows communication between a news server and a news client.
- Web (HTTP): Base protocol for communication on the World Wide Web.

Figure 3-12 shows the relationship of the presentation layer to the session and application layer.

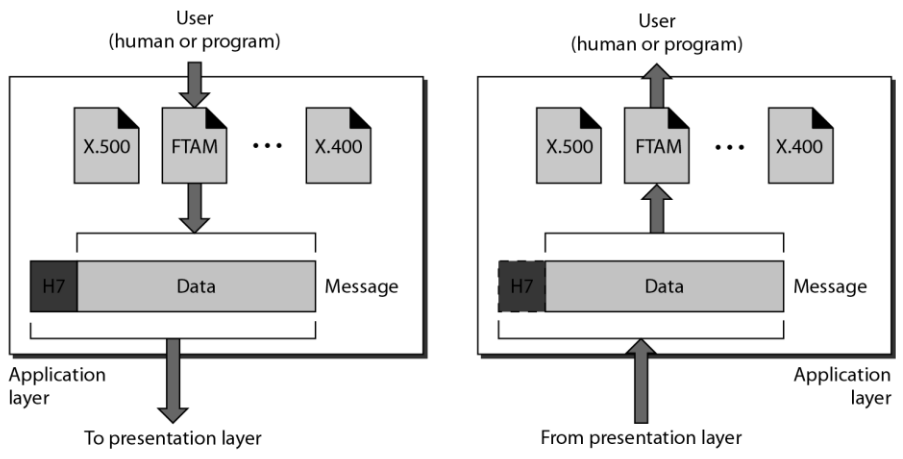


Figure 3.12 *Application layer*

4. Application Layer

Structure

Objectives

- List and explain the various services provided by the Application Layer;
- Describe TCP and UDP services in internet;
- Describe Web and HTTP protocols and formats;
- Discuss the FTP and SMTP protocols.

4.1 Introduction

The **application layer** is the seventh level of the seven-layer OSI model. It interfaces directly to and performs common application services for the application processes; it also issues requests to the layer. The common application layer services provide semantic conversion between associated application processes. *Note:* Examples of common application services of general interest include the virtual file, virtual terminal, and job transfer and manipulation protocols.

In this lesson we study the conceptual and implementation aspects of network applications. We begin by defining key application-layer concepts, including application-layer protocols, clients and servers, processes, sockets, and transport layer interfaces. We then examine several application-layer protocols in detail including HTTP (for the Web), SMTP and POPS (for e-mail), FTP (for file transfer), and DNS (for translating host names to IP addresses).

4-2 Principles of Application-Layer Protocols

A sending process creates and sends messages into the network; a receiving process receives these messages and possibly responds by sending messages back. Networking applications have application-layer protocols that define the format and order of the messages exchanged between processes, as well as defines the actions taken on the transmission or receipt of a message. Figure 4.1 illustrates that processes communicate with each other by using the application layer of the five-layer protocol stack.

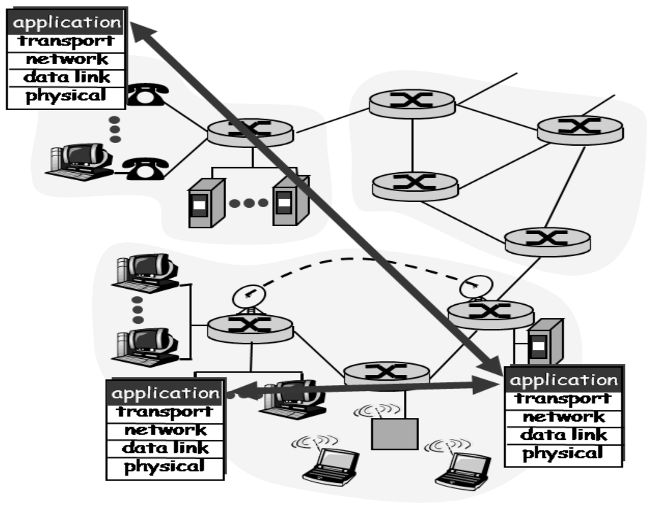


Figure 4-1 Communication for a network application takes place at the application layer

The application layer is a particularly good place to start our study of protocols. It's familiar ground. We're acquainted with many of the applications that rely on the protocols we will study. It will give us a good feel for what protocols are all about and will introduce us to many of the same issues that we'll see again when we study transport, network, and data link layer protocols.

4-2-1 Application-Layer Protocols

It is important to distinguish between **network applications and application-layer protocols.** An application-layer protocol is only one piece (albeit, a big piece) of a network application. Let's look at a Couple of examples. The Web is a network application that allows users to obtain "documents" from Web servers on demand. The Web application consists of many components, including a standard for document formats (that is, HTML), Web browsers (for example, Netscape Navigator and Microsoft Internet Explorer), Web servers (for example, Apache, Microsoft, and Netscape servers), and an application-layer protocol. The Web's application-layer protocol, HTTP (the Hyper-Text Transfer Protocol), defines the format and sequence of the messages that are passed between browser and Web server. Thus, the Web's application-layer protocol, HTTP, is only one piece of the Web application. As another example, consider the Internet electronic mail application. Internet electronic mail also has many components, including: mail servers that house user mailboxes; mail readers that allow users to read and create messages; a standard for defining the structure of an e-mail message; and application-layer protocols that define how messages are passed between servers, how messages are passed between servers and mail readers, and how the contents of certain parts of the mail message (for example, a mail message header) are to be interpreted. The principal application-layer protocol for electronic mail is SMTP (Simple Mail Transfer Protocol). Thus, e-mail's principal application-layer protocol, SMTP, is only one piece (albeit, a big piece) of the e-mail application.

As noted above, an application-layer protocol defines how an application's processes, running on different end systems, pass messages to each other. In particular, an application-layer protocol defines:

• The types of messages exchanged, for example, request messages and response messages

• The syntax of the various message types, such as the fields in the message and how the fields are delineated

• The semantics of the fields, that is, the meaning of the information in the fields

• Rules for determining when and how a process sends messages and responds to messages

4-2-1-1 Client and Server Sides of an Application

As shown in Figure 4.2, a network application typically has two parts or "sides," a **client side** and a **server side.** The client side in one end system communicates with the server side in another end system. For example, a Web browser implements the client side of HTTP, and a Web server implements the server side of HTTP. In e-mail, the sending mail server implements the client side of SMTP, and the receiving mail server implements the server side of SMTP.

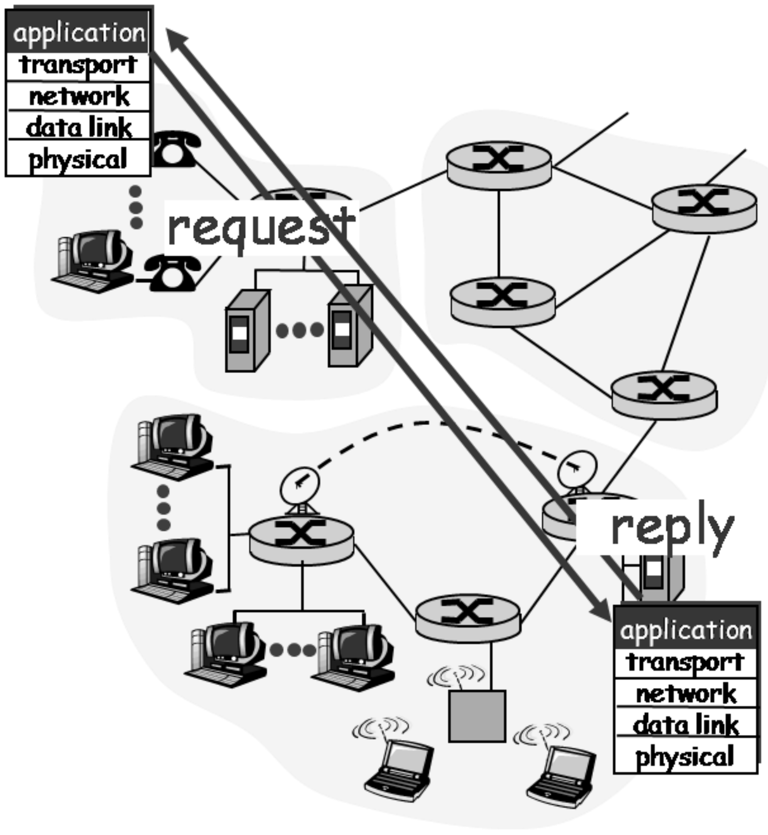


Figure 4-2 Client-server interaction

4-2-1-2 Processes Communicating Across a Network

Many applications involve two processes in two different hosts communicating with each other over a network. The two processes communicate with each other by sending and receiving messages. A process sends messages into, and receives messages from, the network through its *socket.* A process's socket can be thought of as the process's door. When a process wants to send a message to another process on another host, it shoves the message out its door (socket). This sending process assumes that there is a transportation infrastructure on the other side of its door that will transport the message across the Internet to the door of the destination process. Once the message arrives at the destination host, the message passes through the receiving process's door (socket), and the receiving process then acts on the message.

Figure 4.3 illustrates socket communication between two processes that communicate over the Internet. (Figure 4.3 assumes that the underlying transport protocol is TCP, although the UDP protocol could be used as well in the Internet.) As shown in this figure, a **socket** is the interface between the application layer and the transport layer within a host. It is also referred to as the **API (application programmers' interface)** between the application and the network, since the socket is the programming interface with which network applications are built in the Internet. The application developer has control of everything on the application-layer side of the socket but has little control of the transport-layer side of the socket. The only control that the application developer has on the transport-layer side is (1) the choice of transport protocol and (2) perhaps the ability to fix a few transport-layer parameters such as maximum buffer and maximum segment sizes.

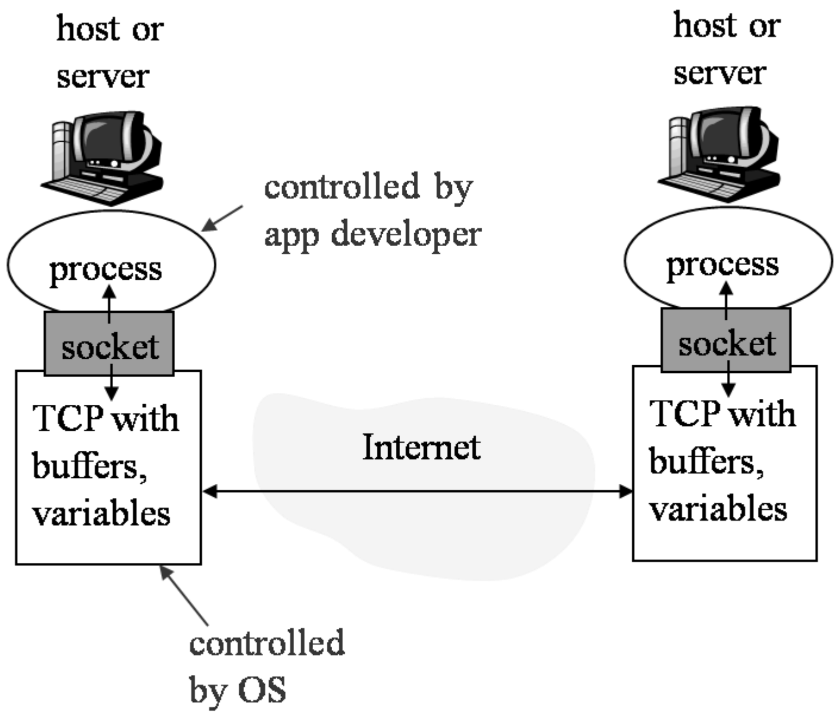

host or server
host or server
controlled by app developer
process
process
socket
socket
TCP with buffers, variables
Internet
TCP with buffers, variables
controlled by OS

Figure 4-3 Application process, sockets, and underlying transport protocol

4-2-1-3 Addressing Processes

In order for a process on one host to send a message to a process on another host, the sending process must identify the receiving process. To identify the receiving process, one must typically specify two pieces of information: (1) the name or address of the host and (2) an identifier that specifies the receiving process in the destination host.

Let us first consider host addresses. In Internet applications, the destination "host is identified by its **IP address**. The address of the host to which a message is destined sending application must also provide information that will allow the receiving hi to direct the message to the appropriate process on that host. A destination **port number** serves this purpose in the Internet.

4-2-2 What Services Does an Application Need?

Interface between the application process and the transport protocol, the application at the sending side sends messages through the door. At the other side of the door, the transport protocol has the responsibility of moving the messages across the network to the door at the receiving process. Many networks, including the Internet, provide more than one transport protocol. When you develop an application, you must choose one of the available transport protocols.

We can broadly classify an application's service requirements along three dimensions: data loss, bandwidth, and timing.

4-2-2-1 Reliable Data Transfer

Some applications, such as electronic mail, instant messaging, file transfer, remote host access. Web document transfers and financial applications require fully reliable data transfer, that is, no data loss. In particular, a loss of file data, or data in a financial transaction, can have devastating consequences (in the latter case, for either the bank or the customer).

4-2-2-2 Bandwidth

Some applications must be able to transmit data at a certain rate in order to be effective. For example, if an Internet telephony application encodes voice at 32 kbps, then it must be able to send data into the network and have data delivered to the receiving application at this rate. If this amount of bandwidth is not available, the application needs to encode at a different rate or it should give up, since receiving half of the needed bandwidth is of no use to such a **bandwidth-sensitive application.**

4-2-2-3 Timing

The final service requirement is that of timing. Interactive real-time application such as Internet telephony, virtual environments, teleconferencing, and multiplayer games require tight timing constraints on data delivery in order to be effective, example, many of these applications require that end-to-end delays be on the o of a few hundred milliseconds or less.

4-2-3 Services Provided by the Internet Transport Protocols

The Internet (and, more generally, TCP/IP networks) makes available two transport protocols to applications, namely, **UDP** (User Datagram Protocol) and **TCP** (Transmission Control Protocol). When a developer creates a new application for the Internet, one of the first decisions that the developer must make is whether to use UDP or TCP. Each of these protocols offers a different service model to the invoking applications.

4-2-3-1 TCP Services

The TCP service model includes a connection-oriented service and a reliable data transfer service. When an application invokes TCP for its transport protocol, 'the application receives both of these services from TCP.

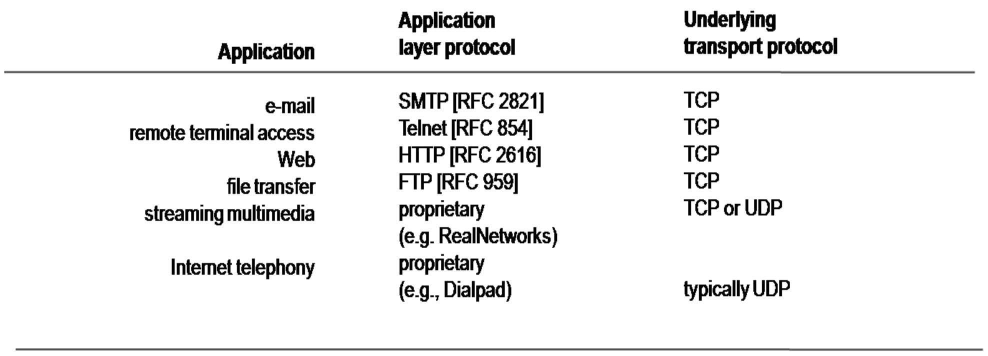
• *Connection-oriented service:* TCP has the client and server exchange transport-layer control information with each other *before* the application-level messages begin to flow. This so-called handshaking procedure alerts the client and server, allowing them to prepare for an onslaught of packets. After the handshaking phase, a **TCP connection** is said to exist between the sockets of the two processes. The connection is a full-duplex connection in that the two processes can send messages to each other over the connection at the same time. When the application is finished sending messages, it must tear down the connection. The service is referred to as a "connection-oriented" service rather than a "connection" service (or a "virtual circuit" service) because the two processes are connected in a very loose manner.

• *Reliable transport service:* The communicating processes can rely on TCP to deliver all data sent without error and in the proper order. When one side of the application passes a stream of bytes into a socket, it can count on TCP to deliver the same stream of data to the receiving socket, with no missing or duplicate bytes.

4-2-3-2 UDP Services

UDP is a no-frills, lightweight transport protocol with a minimalist service model, UDP is connectionless, and so there is no handshaking before the two processes start to communicate. UDP provides an unreliable data transfer service that is, when a process sends a message into a UDP socket, UDP provides *no* guarantee that the message will ever reach the receiving process. Furthermore, messages that do arrive to the receiving process may arrive out of order.

UDP does not include a congestion-control mechanism, so a sending process can pump data into a UDP socket at any rate it pleases (although not all the data may make it to the receiving socket). Because real-time applications usually can tolerate some loss but require a minimal rate, developers of real-time applications often choose to run their applications over UDP. Similar to TCP, UDP provides no guarantee on delay. Figure 4.4 indicates the transport protocols used by some popular Internet applications.



Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Dialpad)	typically UDP

Figure 4-4 popular internet applications, their application-layer protocols, and their underlying transport protocols.

4-3 The Web and HTTP

Until the 1990s the Internet was used primarily by researchers, academics, and university students to log in to remote hosts, to transfer files from local hosts to remote and vice versa, to receive and send news, and to receive and send electronic mail though these applications were (and continue to be) extremely useful, the Internet was essentially unknown outside the academic and research communities. Then, in the early 1990s, a major new application arrived on the scene-the World Wide."The Web is the Internet application that caught the general lie's eye. It dramatically changed how people interact inside and outside work environments. It elevated the Internet from just one of many data networks to essentially the one and data network.

4-3-1 Overview of HTTP

The Hypertext Transfer Protocol (HTTP), the Web's application-layer protocol, is at the heart of the Web. HTTP is implemented in two programs: a client program and a server program. The client program and server program, executing on different end systems, talk to each other by exchanging-HTTP messages. HTTP defines the structure of these messages and how the client and server exchange the messages. Before explaining HTTP in detail, it is useful to review some Web terminology.

A Web page (also called a document) consists of objects. An **object** is simply a file such as an HTML file, a JPEG image, a GIF images, a Java applet, an audio clip, and so on that is addressable by a single URL. Most Web pages consist of a base **HTML file** and several referenced objects. For example, if a Web page contains HTML text and five JPEG images, then the Web page has six objects: the base HTML file plus the five images. The base HTML file references the other objects in the page with the objects' URLs. Each URL has two components: the host name of the server that houses the object and the object's path name. For example, the URL

www. someSchool.edu/someDepartment/picture.gif

has www.someSchool.edu for a host name and /someDepartment/ picture. gif for a path name. A **browser** is a user agent for the Web; it displays the requested Web page and provides numerous navigational and configuration features. Web browsers also implement the client side of HTTP. Thus, in the context of the Web, we will interchangeably use the words "browser" and "client." Popular Web browsers include Netscape Communicator and Microsoft Internet Explorer.

A Web server houses Web objects, each addressable by a URL. Web servers also implement the server side of HTTP. Popular Web servers include Apache and Microsoft Internet Information Server. (Netcraft provides a nice survey of Web sen penetration [Netcraft 2002].)

HTTP defines how Web clients (for example, browsers) request Web pages from the Web and how servers transfer Web pages to clients. We discuss the interaction between client and server in detail below, but the general idea is illustrated in Figure 5.5. When a user requests a Web page (for example, clicks on a hyperlink), the browser sends HTTP request messages for the objects in the page to the server. The (server receives the requests and responds with HTTP response messages that contain the objects.

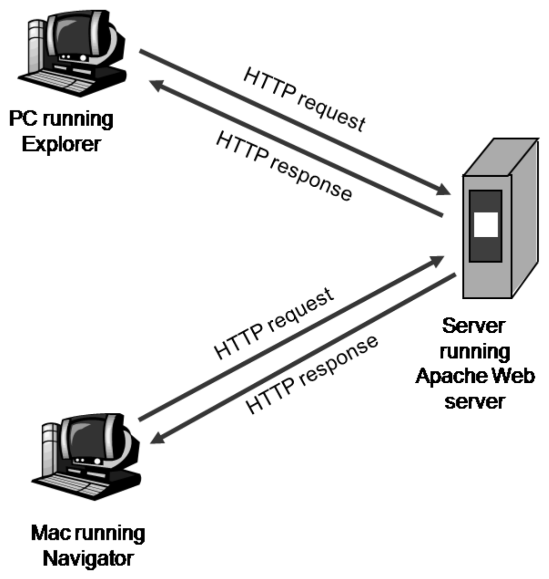


Figure 4-5 HTTP request-response behavior

4-3-2 Non-persistent and Persistent Connections

HTTP can use both non-persistent connections and persistent connections. Non-persistent connections only a single Web object is transferred over a TCP connection. Persistent connections act as default mode. We'll see that with persistent connections, a new connection need not be set up for the transfer of each Web object; instead, multiple Web objects can be transferred during the lifetime of a single connection.

Non-persistent Connections because each TCP connection closed after the server sends the object the connection does not persist for other objects. Note that each TCP connection transports exactly one request message a one response message.

Persistent Connections have some shortcomings. First, a brand-new connection must be established and maintained for each requested object. For each of these connections, TCP buffers must be allocated and TCP variables must be kept in both the client and server. This can place a serious burden on the Web server, which may be serving requests from hundreds of different clients simultaneously. Second, as we just described, each object suffers a delivery delay of two RTTs—one RTT to establish the TCP connection and one RTT to request and receive an object.

4-3-3 HTTP Message Format

The HTTP specifications include the definitions of the HTTP message formats. There are two types of HTTP messages, request messages and response messages, both of which are discussed below.

HTTP Request Message consists of a request line, headers, and sometimes a body as shown figure 4-6.

HTTP Response Message consists of a status line, headers, and sometimes a body as shown figure 4-6.

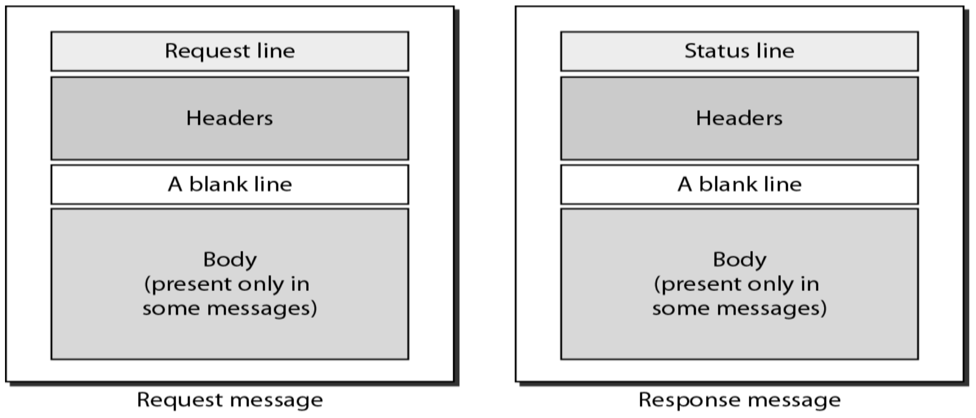


Figure 4-6 general formats of request and response messages

4-4 File transfer: FTP

Given a reliable end-to-end transport protocol like TCP, File Transfer might seem trivial. But, the details authorization, representation among heterogeneous machines makes the protocol complex.

FTP offers many facilities:

- Interactive Access: Most implementations provide an interactive interface that allows humans to easily interact with remote servers.
- Format (representation) specification: FTP allows the client to specify the type and format of stored data.
- Authentication Control: FTP requires client to authorize themselves by sending a login name and password to the server before requesting file transfers.

FTP Process Model

FTP allows concurrent accesses by multiple clients. Clients use TCP to connect to the server. A master server awaits connections and creates a slave process to handle each connection. Unlike most servers, the slave process does not perform all the necessary computation. Instead the slave accepts and handles the control connection from the client, but uses an additional process to handle a separate data transfer connection. The control connection carries the command that tells the server which file to transfer.

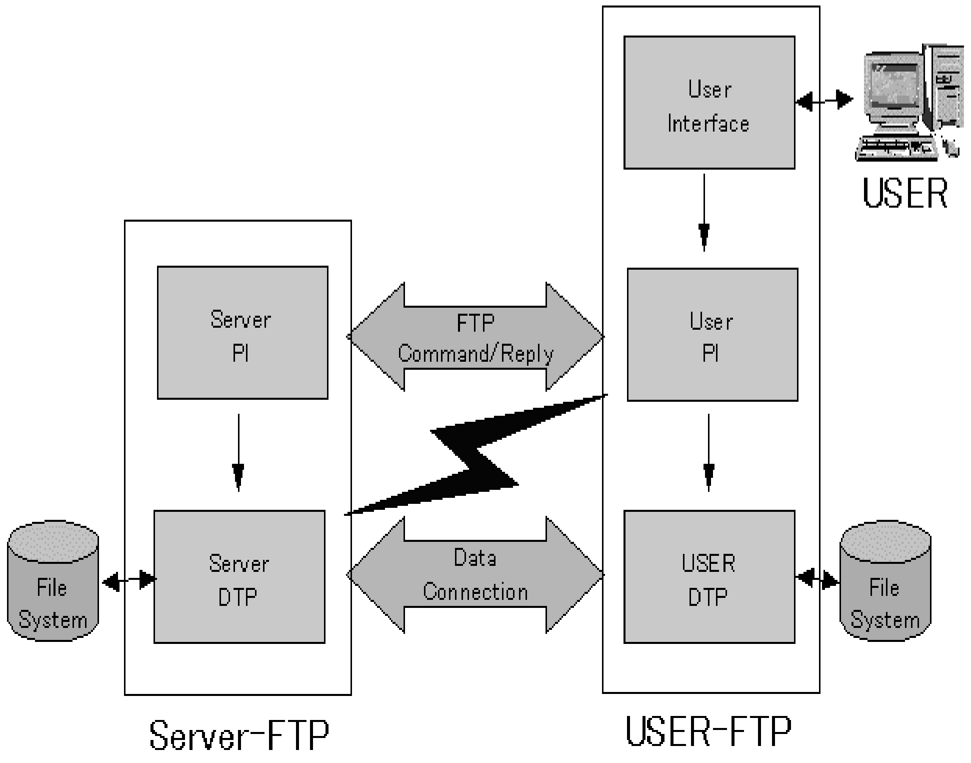


Figure 4-7 FTP moves files between local and remote file systems

Data transfer connections and the data transfer processes that use them can be created dynamically when needed, but the control connection persists throughout a session. Once the control connection disappears, the session is terminated and the software at both ends terminates all data transfer processes.

In addition to passing user commands to the server, FTP uses the control connection to allow client and server processes to coordinate their use of dynamically assigned TCP protocol ports and the creation of data transfer processes that use those ports.

Proxy commands - allows one to copy files from any machine to any other arbitrary machine. The machine the files are being copied to need not be the client but any other machine. Sometimes some **special processing** can be done which is not part of the protocol. For example if a request for copying a file is made by issuing command 'get file_A.gz' and the zipped file does not exist but the file file-A does, then the file is automatically zipped and sent. Consider what happens when the **connection breaks during a FTP**

session. Two things may happen, certain FTP servers may again restart from the beginning and whatever portion of the file had been copied is overwritten. Other FTP servers may ask the client how much it has already read and it simply continues from that point.

4-5 EMAIL (electronic mail - SMTP, MIME, ESMTP)

Email is the most widely used application service which is used by computer users. It differs from other uses of the networks as network protocols send packets directly to destinations using timeout and retransmission for individual segments if no ack returns. However in the case of email the system must provide for instances when the remote machine or the network connection has failed and take some special action. Email applications involve two aspects

- User-agent (pine, elm etc.)
- Transfer agent (sendmail daemon etc.)

When an email is sent it is the mail transfer agent (MTA) of the source that contacts the MTA of the destination. The protocol used by the MTA's on the source and destination side is called SMTP. SMTP stands for **Simple Mail Transfer Protocol.**. There are some protocols that come between the user agent and the MTA eg. POP, IMAP which are discussed later.

4-5-1Mail Gateways

Mail gateways are also called mail relays, mail bridges and in such systems the sender's machine does not contact the receiver's machine directly but sends mail across one or more intermediate machines that forward it on. These **intermediate machines** are called mail gateways. Mail gateways are introducing unreliability. Once the sender sends to first intermediate m/c then it discards its local copy. So failure at an intermediate machine may result in message loss without informing the sender or the receiver. Mail gateways also introduce delays. Neither the sender nor the receiver can determine how long the delay will last or where it has been delayed.

However mail gateways have an advantage providing interoperability i.e. they provide connections among standard TCP/IP mail systems and other mail systems as well as between TCP/IP internets and networks that do not support Internet protocols. So when there is a change in protocol then the mail gateway helps in translating the mail message from one protocol to another since it will be designed to understand both.

4-5-2 Simple Mail Transfer Protocol (SMTP)

TCP/IP protocol suite specifies a standard for the exchange of mail between machines. It was derived from the (MTP) Mail Transfer Protocol. it deals with how the underlying mail delivery system passes messages across a link from one machine to another. The mail is enclosed in what is called an **envelope.** The envelope contains to and from fields and these are followed by the mail. The mail consists of two parts namely the Header and the Data.

The Header has to and from fields. If Headers are defined by us they should start with X. The standard headers do not start with X. In SMTP data portion can contain only printable ASCII characters The old method of sending a binary file was to send it in uuencoded form but there was no way to distinguish between the many types of binary files possible eg: tar, gz , dvi etc.

4-5-3 MIME (Multipurpose Internet Mail Extension)

This allows the transmission of Non ASCII data through the email, MIME allows arbitrary data to be encoded in ASCII and sent in a standard email message. Each MIME message includes information that tells the recipient the type of data and the type of encoding used and this information along with the MIME version resides in the MIME header. Typical MIME header looks like

MIME-Version: 1.0
Content-Description:
Content-Id:
Content-Type: image/gif
Content-Transfer-Encoding: base64

Content Description: contains the file name of the file that is being sent. Content type is an important field that specifies the data format i.e. tells what kind of data is being sent. It contains two identifiers a content type and a subtype separated by a slash. For e.g. image/gif.

There are 7 Content Types

1. text
2. image
3. video
4. audio
5. application
6. multipart
7. message

Content type **message**

it supports 3 subtypes namely

1. RFC822 - the old mail message format
2. Partial means that ordinary message is just a part and the receiver should wait for all the parts before putting it in the mailbox.
3. External body - destination MTA will fetch file from remote site.

Content Type - **Multipart**
multiple messages which may have different content types can be sent together. It supports 4 subtypes namely

1. Mixed -Look at each part independently
2. Alternative - The same message is sent in multiple types and formats and the receiver may choose to read the message in any form he wishes.
3. Parallel -The different parts of the message have to be read in parallel. i.e. audio , video and text need to be read in a synchronized fashion
4. Digest -There are multiple RFC messages in mail. The addresses of the receivers are in the form of a mailing list. Although file header is long it prevents cluttering of mail box.

4-5-4 PROBLEMS WITH SMTP

1. There is no convenient way to send nonprintable characters
2. There is no way to know if one has received mail or not or has read it or not.
3. Someone else can send a mail on my behalf.

So a better protocol was proposed ESMTP stands for Extended Simple Mail Transfer Protocol. It is compatible with SMTP. Just as the first packet sent in SMTP is HELO similarly in ESMTP the first packet is called EHELO. If the receiver supports ESMTP then it will answer to this EHELO packet by sending what data type and what kind of encoding it supports. Even a SMTP based receiver can reply to it. Also if there is an error message or there is no answer then the sender uses SMTP.

4-5-6 Delivery Protocols

The delivery protocols determine how the mail is transferred by the mail transfer agent to the user agent which provides an interface for reading mails. There are 3 kinds.

1. POP3 (Post Office Protocol) Here the mail person accesses the mail box from say a PC and the mail gets accumulated on a server. So in POP3 the mail is downloaded to the PC at a time interval which can be specified by the user. POP3 is used when the mail is always read from the same machine, so it helps to download the mail to it in advance.

2. IMAP (Intermediate Mail Access Protocol) Here the user may access the mail box on the server from different machines so there is no point in downloading the mail before hand. Instead when the mail has to be read one has to log on to the server. (IMAP thus provides **authentication**) The mailbox on the server can be looked upon as a **relational database.** **3. DMSP (Distributive Mail System Protocol)** there are multiple mailboxes on different servers. To read the mail I connect to them from time to time and whenever I do so the mail will be downloaded. When a reply is sent then it will put the message in a queue. Thus DMSP is like a **pseudo MTA.**

Ensuring Network Security

1. How to ensure that nobody else reads your mail?
2. How to be sure that the mail has not been seen by someone else in your name?
3. Integrity i.e. mail has not been tampered with
4. Non-Reputability- means once I send a mail I cannot deny it, and this fact can be proved to a third person
5. Authentication

Mechanisms (PGP & PEM)

PGP (Pretty Good Privacy) - It uses some cryptography algorithm to crypt the messages.

Symmetric PGP- The key used for encryption and decryption is the same. **Asymmetric PGP -** The key used for encryption and decryption is different keys come in pairs public (known to all) and private. Which everybody ha usually encryption is done using public key so that the private key is used for decryption by the receiver only for whom the message is meant. E.g. of Symmetric PGP is DES, IDEA

E.g. of Asymmetric PGP is RSA

Symmetric is usually faster in asymmetric PGP there is a problem of key distribution. A hash function is applied on every message so that no two messages hash to the same value. Now the hash function is encrypted. If the hash function of source and destination matches then No tampering. If the key for encryption is private then not everybody can generate the message although anyone can read it. So this scheme **lacks privacy** tackles the other security issues.

Summary

In this unit we've covered a *tremendous* amount of material! In the first and second part of this introductory unit we've looked at the various pieces of hardware and software that make up the Internet in particular, and computer networks in general. We started at the "edge" of the network, looking at end systems and applications, and at the transport service provided to the applications running on the end systems. Using network-based distributed applications as examples, we introduced the notion of a protocol - a key concept in networking. We then dove deeper inside the network, into the network core, identifying packet-switching and circuit switching as the two basic approaches for transporting data through a telecommunication network, and examining the strengths and weaknesses of each approach. We then looked at the lowest (from an architectural standpoint) parts of the network the link layer technologies and physical media typically found in the access network.

In the third part of this introductory unit we then took the broader view on networking. From a performance standpoint, we identified the causes of packet delay and packet loss in the Internet. We identified key architectural principles (layering, service models) in networking. We then examined the structure of today's Internet. We finished our introduction to networking with a brief history of computer networking. The first chapter in itself constitutes a mini-course in computer networking.

In fourth part of this unit studied both the conceptual and the implementation aspects of network applications. We've learned about the ubiquitous client-server paradigm adopted by Internet applications and seen its use in the HTTP, FTP, SMTP, POP3 and DNS protocols. We've studied these important application-level protocols, and their associated applications (the Web, file transfer, e-mail, and the domain name system) in some detail. We've examined how the socket API can be used to build network applications and walked through not only the use of sockets over connection-oriented (TCP) and connectionless (UDP) end-to-end transport services, but also built a simple web server using this API. The first step in our top-down journey "down" the layered network architecture is complete.

So, we have indeed covered a tremendous amount of ground in this first chapter! If you're a bit overwhelmed, don't worry. In the following chapters we will revisit all of these ideas, covering them in much more detail (that's a promise, not a threat!). At this point, we hope you leave this chapter with a still-developing intuition for the pieces that make up a network, a still-developing command for the vocabulary of networking (don't be shy to refer back to this chapter), and an ever-growing desire to learn more about networking. That's the task ahead of us for the rest of this book

Exercise

1) What are the two types of services that the Internet provides to its applications? What are some of characteristics of each of these services?

2) It has been said that flow control and congestion control are equivalent. Is this true for the Internet's connection-oriented service? Are the objectives of flow control and congestion control the same?

3) Briefly describe how the Internet's connection-oriented service provides reliable transport.

4) What advantage does a circuit-switched network have over a packet-switched network?

4) What advantages does TDM have over FDM in a circuit-switched network?

5) Suppose that between a sending host and a receiving host there is exactly one packet switch. The transmission rates between the sending host and the switch and between the switch and the receiving host are R_1 and R_2, respectively. Assuming that the router uses store-and-forward packet switching, what is the total end-to-end delay to send a packet of length L. (Ignore queuing and propagation delay?)

6) What are some of the networking technologies that use virtual circuits? Find good URLs that discuss and explain these technologies.

7) What is meant by connection state information in a virtual-circuit network?

8) Suppose you are developing a standard for a new type of network. You need to decide whether your network will use VCs or datagram routing. What are the pros and cons for using VCs?

9) List five tasks that a layer can perform. It is possible that one (or more) of these tasks could be performed by two (or more) layers?

10) What are the five layers in the Internet protocol stack? What are the principle responsibilities for each of these layers?

11) Which layers in the Internet protocol stack does a router process?

12) List five non-proprietary Internet applications and the application-layer protocols that they use.

13) For a communication session between two hosts, which host is the client and which is the server?

14) What information is used by a process running on one host to identify a process running on another host?

15) List the various network-application user agents that you use on a daily basis.

16) What is meant by a handshaking protocol?

17) Why do HTTP, FTP, SMTP, POP3 and IMAP run on top of TCP rather than UDP?

18) Why is it said that FTP sends control information "out of band"?

UNIT – II

1. Transport Layer

Objectives

- List and explain the various services provided by the Transport Layer;
- Establish and release TCP connections;
- Describe TCP and UDP header formats;
- Describe TCP Flow Control mechanism and how it is different from data link layer, and

1-1 Introduction

The transport layer of a TCP/IP computer network is situated above the network layer and below the applications layer. The network layer provides unreliable packet transfer service between any two hosts. The transport layer uses this network service and provides transport services between any two applications in the network. Applications include email (SMTP), remote login (TELNET, SSH), file transfer (FTP), web browsers (HTTP), remote file systems (NFS), name-to-address translation (DNS), voice and video streaming (e.g. Real Networks), internet telephony (e.g. Vocaltec), etc. We refer to applications using the transport service as transport users, or users for short. The transport services are provided by transport protocols, which are distributed algorithms running on the hosts. There are different transport services, and hence different transport protocols. We refer to the components of the protocols running at the hosts as transport entities, or entities for short.

1-2 Transport-Layer Services

A transport-layer protocol provides for **logical communication** between application processes running on different hosts. By logical communication, we mean that from an application's perspective, it is as if the hosts running the processes were directly connected; in reality, the hosts may be on opposite sides of the planet, connected via numerous routers and a wide range of link types. Application processes use the logical communication provided by the transport layer to send messages to each other, free from the worry of the details of the physical infrastructure used to carry these messages. Figure 1.1 illustrates the notion of logical communication.

As shown in Figure 4.1, transport-layer protocols are implemented in the end systems but not in network routers. Network routers act only on the network-layer fields of the layer-3 PDUs (that is, network-layer protocol data units); they do not act on the transport-layer fields. On the sending side, the transport layer converts the messages it receives from a sending application process into 4-PDUs (that is, transport-layer protocol data units). This is done by (possibly) breaking the application messages into smaller chunks and adding a transport-layer header to each chunk to create 4-PDUs. The transport layer then passes the 4-PDUs to the network layer at the sending end system, where each 4-PDU is encapsulated into a 3-PDU. On the receiving side, the transport layer receives the 4-PDUs from the network layer below, removes the transport header from the 4-PDUs, reassembles the messages, and passes them to the receiving application process.

More than one transport-layer protocol may be available to network applications. For example, the Internet has two protocols TCP and UDP. Each of these protocols provides a different set of transport-layer services to the invoking application.

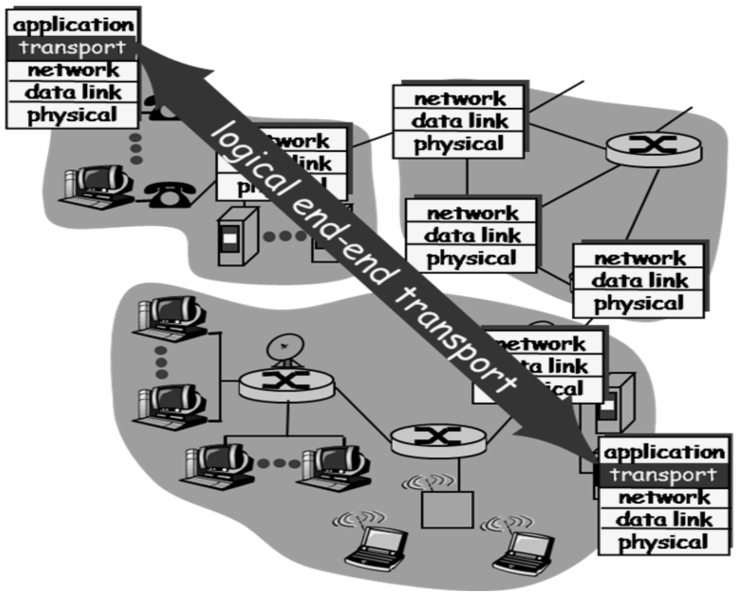


Figure 1-1 Transport layer provides logical rather than physical communication between application processes

1-2-1 Relationship between Transport and Network Layers

Transport layer lies just above the network layer in the protocol stack. Whereas a transport layer protocol provides *logical communication between processes* running on different hosts, a network-layer protocol provides *logical communication between hosts.* This distinction is subtle but important. Let's examine this distinction with the aid of a household analogy.

Consider two houses, one on the East Coast and the other on the West Coast with each house being home to a dozen kids. The kids in the East Coast household are cousins of the kids in the West Coast household. The kids in the two households love to write to each other each kid writes each cousin every week, with each letter delivered by the traditional postal service in a separate envelope. Thus, each household sends 144 letters to the other household every week. (These kids would save a lot of money if they had e-mail!) In each of the households there is one kid Ann in the West Coast house and Bill in the East Coast house responsible for mail collection and mail distribution. Each week Ann visits all her brothers and sisters, collects the mail, and gives the mail to a postal-service mail carrier who makes daily visits to the house. When letters arrive at the West Coast house, Ann also has the job of distributing the mail to her brothers and sisters. Bill has a similar job on the East Coast.

In this example, the postal service provides logical communication between the two houses the postal service moves mail from house to house, not from person to person. On the other hand, Ann and Bill provide logical communication among the cousins Ann and Bill pick up mail from, and deliver mail to, their brothers and sisters. Note that from the cousins' perspective, Ann and Bill *are* the mail service, even though Ann and Bill are only a part (the end-system part) of the end-to-end delivery process. This household example serves as a nice analogy for explaining how the transport layer relates to the network layer:

Application messages = letters in envelopes

Processes = cousins

Hosts (also called end systems) = houses

Transport-layer protocol = Ann and Bill

Network-layer protocol = postal service (including mail carriers)

Continuing with this analogy, note that Ann and Bill do all their work within their respective homes; they are not involved, for example, in sorting mail in any intermediate mail center or in moving mail from one mail center to another. Similarly, transport-layer protocols live in the end systems. Within an end system, a transport protocol moves messages from application processes to the network edge (that is, the network layer) and vice versa, but it doesn't have any say about how the messages are moved within the network core. In fact, as illustrated in Figure 1.1, intermediate routers neither act on, nor recognize, any information that the transport I layer may have added to the application messages.

Continuing with our family saga, suppose now that when Ann and Bill go on vacation, another cousin-pair-say, Susan and Harvey-substitute for them and provide the household-internal collection and delivery of mail. Unfortunately for the two families, Susan and Harvey do not do the collection and delivery in exactly the same way as Ann and Bill. Being younger kids, Susan and Harvey pick up and drop off the mail less frequently and occasionally lose letters (which are sometimes chewed up by the family dog). Thus, the cousin-pair Susan and Harvey do not provide the same |||et *of* services (that is, the same service model) as Ann and Bill. In an analogous manner, a computer network may make available multiple transport protocols, with each protocol offering a different service model to applications.

The possible services that Ann and Bill can provide are clearly constrained f the possible services that the postal service provides. For example, if the postal’s vice doesn't provide a maximum bound on how long it can take to deliver mail between the two houses (for example, three days), then there is no way that Ann and Bill can guarantee a maximum delay for mail delivery between any of the cousin pairs. In a similar manner, the services that a transport protocol can provide area ten constrained by the service model of the underlying network-layer protocol. If the network-layer protocol cannot provide delay or bandwidth guarantees for 4-PDUs sent between hosts, then the transport-layer protocol cannot provide delay or bandwidth guarantees for messages sent between processes.

1-2-2 Overview of the Transport Layer in the Internet

Internet, and more generally a TCP/IP, network, makes available two distinct transport-layer protocols to the application layer. One of these protocols is **UDP** (User Datagram Protocol), which provides an unreliable, connectionless vice to the invoking application. The second of these protocols is **TCP** (Transmission Control Protocol), which provides a reliable, connection-oriented service to the invoking application. When designing a network application, the application developer must specify one of these two transport protocols.

To simplify terminology, when in an Internet context, we refer to the 4-PDU as a **segment.** We mention, however, that the Internet literature (for example, the RFCs) also refers to the PDU for TCP as a segment but often refers to the PDU for UDP as a datagram. But this same Internet literature also uses the terminology datagram for the network-layer PDU! For an introductory book on computer networking such as this, we believe that it is less confusing to refer to both TCP and UDP PDUs as segments, and reserve the terminology datagram for the network-layer PDU.

Before proceeding with our brief introduction of UDP and TCP, it is useful to say a few words about the Internet's network layer. The Internet's network-layer protocol has a name IP, for Internet Protocol. IP provides logical communication between hosts. The IP service model is a **best-effort delivery service.** This means that IP makes its "b to deliver segments between communicating hosts, *but it makes no guarantees .*In particular, it does not guarantee segment delivery, it does not guarantee orderly delivery of segments, and it does not guarantee the integrity of the data in the segments. For these reasons, IP is said to be an **unreliable service.** We also mention here that every host has at least one network-layer address, a so-called IP address. We will examine IP addressing in detail in Chapter 4; for this chapter we need only keep in mind that *each host has an IP address.*

Having taken a glimpse at the IP service model, let's now summarize the service models provided by UDP and TCP. The most fundamental responsibility of UDP and TCP is to extend IP's delivery service between two end systems to a delivery service between two processes running on the end systems. Extending host-to-host delivery to process-to-process delivery is called **transport-layer multiplexing** and **demultiplexing.** UDP and TCP also provide integrity checking by including error-detection fields in their headers. These two minimal transport-layer services process-to-process data delivery and error checking are the only two services that UDP provides! In particular, like IP, UDP is an unreliable service it does not guarantee that data sent by one process will arrive intact (or at all!) to the destination process.

TCP, on the other hand, offers several additional services to applications. First and foremost, it provides **reliable data transfer.** Using flow control, sequence numbers, acknowledgments, and timers (techniques we'll explore in detail in this chapter), TCP ensures that data is delivered from sending process to receiving process, correctly and in order. TCP thus converts IP's unreliable service between end systems into a reliable data transport service between processes. TCP also provides ' congestion **control.** Congestion control is not so much a service provided to the invoking application as it is a service for the Internet as a whole, a service for the ; general good. Loosely speaking, TCP congestion control prevents any one TCP connection from swamping the links and switches between communicating hosts with an excessive amount of traffic. In principle, TCP permits TCP connections traversing a congested network link to equally share that link's bandwidth. This is done by regulating the rate at which the sending-side TCPs can send traffic into the net-I work. UDP traffic, on the other hand, is unregulated. An application using UDP I transport can send at any rate it pleases, for as long as it pleases.

1-3 Multiplexing and Demultiplexing

In this section we discuss transport-layer multiplexing and demultiplexing extending the host-to-host delivery service provided by the network layer to a process-to-process delivery service for applications running on the host keep the discussion concrete, we'll discuss this basic transport-layers context of the Internet. We emphasize, however, that a multiplexing/de service is needed for all computer networks.

At the destination host, the transport layer receives segments (that is, transport layer PDUs) from the network layer just below. The transport layer has the responsibility of delivering the data in these segments to the appropriate application process running in the host. Let's take a look at an example. Suppose you are sitting in front of your computer, and you are downloading Web pages while running one FTP session and two Telnet sessions. You therefore have four network application process running two Telnet processes, one FTP process, and one HTTP process. When the transport layer in your computer receives data from the network layer below, it needs to direct received data to one of these four processes. Let's now examine how this is done.

As shown in Figure 1.2, the transport layer in the receiving host does not actually deliver data directly to a process, but instead to an intermediary **socket.** Because at any given time there can be more than one socket in the receiving host, each socket has a unique identifier. The format of the identifier depends on whether the socket is a UDP or a TCP socket, as we shall discuss shortly.

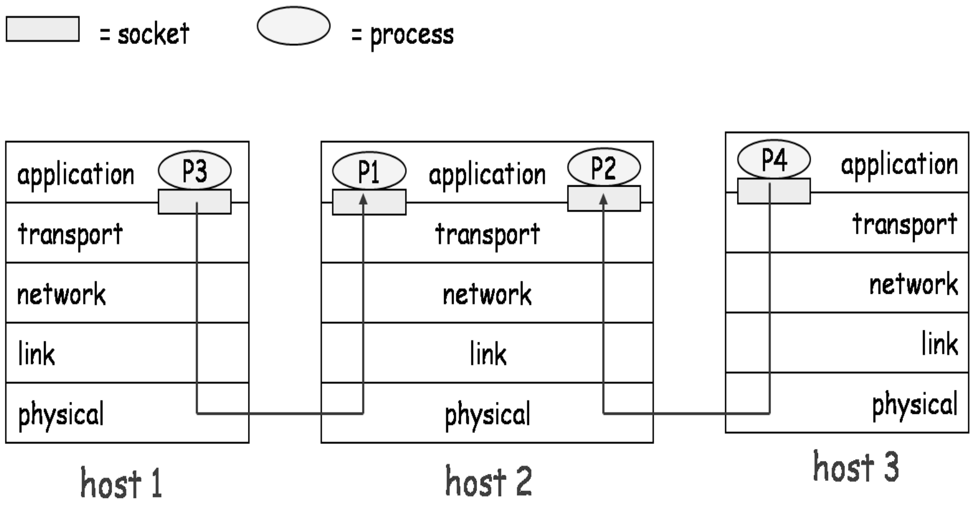


Figure 1-2 Transport layer multiplexing and demultiplexing

Now let's consider how a receiving host directs an incoming transport-layer segment to the appropriate socket. Each transport-layer segment has a set of fields in the segment for this purpose. At the receiving end, the transport layer examines these fields to identify the receiving socket and then directs the segment to that socket. This job of delivering the data in a transport-layer segment to the correct socket is called **demultiplexing.** The job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information (that will later be used in demultiplexing) to create segments, and passing the segments to the network layer is called **multiplexing.** Note that the transport layer in the middle host in Figure 1.2 must demultiplex segments arriving from the network layer below to either process P_1 or P_2, above; this is done by directing the arriving segment's data to the corresponding process's socket. The transport layer in the middle host must also gather outgoing data from these sockets, form transport-layer segments, and pass these segments down to the network layer.

To illustrate the demultiplexing job, recall the household metaphor in the previous section. Each of the kids is identified by his or her name. When Bill receives a batch of mail from the mail carrier, he performs a demultiplexing operation by observing to whom the letters are addressed and then hand-delivering the mail to his brothers and sisters. Ann performs a multiplexing operation when she collects letters from her brothers and sisters and gives the collected mail to the mail person.

Now that we understand the roles of transport-layer multiplexing and demultiplexing, let us examine how it is actually done in a host. From the discussion above, we know that transport-layer multiplexing requires (1) that sockets have unique identifiers and (2) that each segment have special fields that indicate the socket to which the segment is to be delivered. These special fields, illustrated in Figure 1.3, are the **source port number field** and the **destination port number field.** Each port number is a 16-bit number, ranging from 0 to 65535. The port numbers ranging from 0 to 1023 are called **well-known port numbers** and are restricted, which means that they are reserved for use by well-known application protocols such as HTTP (which uses port number 80) and FTP (which uses port number 21). The list of well-known port numbers is given in RFC 1700 and is updated at http://www.iana.org [RFC 3232].

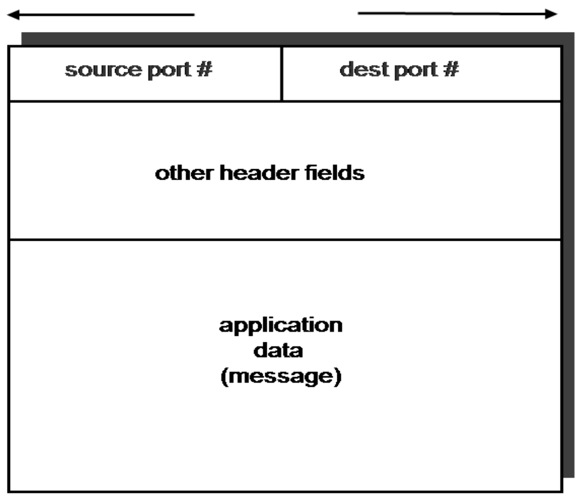


Figure 1-3 Source and destination port number fields in a transport layer segment

It should now be clear how the transport layer *could* implement the demultiplexing service: Each socket in the host can be assigned a port number, and when a segment arrives to the host, the transport layer examines the destination port number in the segment and directs the segment to the corresponding socket. The segment's data then passes through the socket into the attached process. As we shall see, this is basically how UDP does it. However, as we shall also see, multiplexing/demultiplexing in TCP is yet more subtle.

1-3-1 Connectionless Multiplexing and Demultiplexing

The Java program running in a host can create socket with the line:

```
DatagramSocket mySocket = new DatagramSocket();
```

When a UDP socket is created in this manner, the transport layer automatically assigns a port number to the socket. In particular, the transport layer assigns a port nu the range 1024 to 65535 that is currently not being used by any other UDP port in the host. Alternatively, a Java program could create a socket with the line:

```
DatagramSocket mySocket = new DatagramSocket(19157)
```

In this case, the application assigns a specific socket number namely, 19157 to the UDP socket. If the application developer writing the code is implementing the server side of a "well-known protocol," then the developer would have to assign the corresponding well-known port number. Typically, the client side of the application lets the transport layer automatically (and transparently) assign the port numbers, whereas the server side of the application assigns a specific port number.

With port numbers assigned to UDP sockets, we can now precisely describe UDP multiplexing/demultiplexing. Suppose in host-A process with UDP socket 19157 wants to send a chunk of application data to a process with UDP socket 46428 in host-B. The transport layer in host-A creates a transport-layer segment that includes the application data, the source port number (19157), the destination port number (46428), and two other values. The transport layer then passes the resulting segment network layer. The network layer encapsulates the segment in an IP datagram and makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the receiving host B, the receiving host examines the destination port number in the segment (46428) and delivers the segment to its socket identified by port 46428. Note that host B could be running multiple processes, each with its own UDP socket and associated port number. As UDP segments arrive from the network, host B directs (demultiplexes) each segment to the appropriate socket by examining the segment's destination port number.

It is important to note that a UDP socket is fully identified by a two-tuple consisting of a destination IP address and a destination port number.

As a consequence, if two UDP segments have different source IP addresses and/or source port numbers, but have the same *destination* IP address and *destination* port number, then the two segments will be directed to the same destination process via the same destination socket.

1-3-2 Connection-Oriented Multiplexing and Demultiplexing

In order to understand TCP demultiplexing, we have to take a close look at TCP sockets and TCP connection establishment. One subtle difference between socket and a UDP socket is that a TCP socket is identified by a 4-tuple: (source IP address, source port number, destination IP address, destination port number). Thus, when a TCP segment arrives from the network to a host, the host uses all four values to direct (demultiplex) the segment to the appropriate socket. In particular contrast with UDP, two arriving TCP segments with different source IP addresses or source port numbers will be directed to two different sockets.

• The TCP server application has a "welcoming socket," which waits for co establishment requests from TCP clients on port number 6789.

• The TCP client generates a connection-establishment segment with the line:

```
Socket clientSocket = new Socket ("serverHostName", 6789);
```

• A connection-establishment request is nothing more than a TCP segment with destination port number 6789 and a special connection-establishment the TCP header. The segment also includes a source port number, which was chosen by the client. The line above also creates a TCP socket for the client process, through which data can enter and leave the live the client process.

• When the server receives the incoming connection-request segment from a client, it informs the server process, which creates a connection socket:

```
Socket connectionSocket = welcomeSocket.accept();
```

• Also, the server notes the following four values in the connection-request segment: (1) the source port number in the segment, (2) the IP address of the source host, (3) the destination port number in the segment, and (4) its own IP address. The newly created connection socket is identified by these four values; all subsequently arriving segments whose source port, source IP address, destination port, and destination IP address match these four values will be demultiplexed to this socket. With the TCP connection now in place, the client and server can now send data to each other.

The server host may support many simultaneous TCP sockets, with each sockets attached to a process, and with each socket identified by its own 4-tuple. When a TCP segment arrives to the host, all four fields (source IP address, source port, destination IP address, destination port) are used to direct (demultiplex) the segment to the appropriate socket.

The situation is illustrated in Figure 1.4, in which host C initiates two HTTP sessions to server B, and host A initiates one HTTP session to B. Hosts A and C and server B each have their own unique IP address A, C, and B, respectively. Host C assigns two different source port numbers (26145 and 7532) to its two HTTP connections. Because host A is choosing source port numbers independently of *C,* it might also assign a source port of 26145 to its HTTP connection. Nevertheless, server B will still be able to demultiplex correctly the two connections having S same source port number, since the two connections have different source IP addresses.

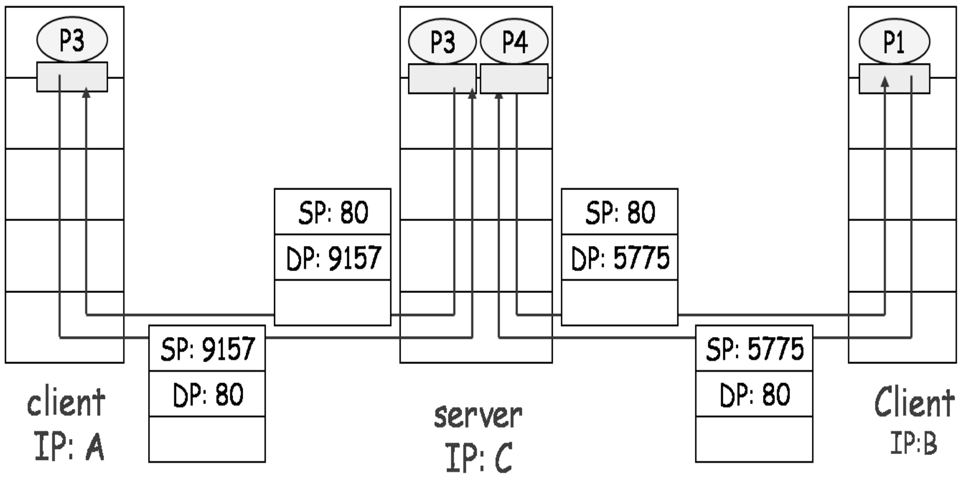


Figure 1-4 two clients, using the same destination port number(80) to communicate with the same web server application

1-3-3 Web Servers and TCP

Before closing this discussion, it is instructive to say a few additional words about web servers and how they use port numbers. Consider a host running a Web server, such as an Apache Web server, on port 80. When clients (for example, browsers) send segments to the server, *all* segments will have destination port 80. In particular, both the initial connection-establishment segments and the segments carrying HTTP request messages will have destination port 80. As we have just described, the server distinguishes the segments from the different clients by the source IP addresses and source port numbers.

Web servers typically spawn a new process or create a new thread for each new client connection. Figure 1.4 shows a Web server that spawns a new process for each connection. As shown in Figure 1-4, each of these processes has its own connection socket through which HTTP requests arrive and HTTP responses are sent. We mention, however, that there is not always a one-to-one correspondence between connection sockets and processes. In fact, today's high-performing Web servers often use only one process, but create a new connection socket fore new client connection. For such a server, at any given time there may be many connection sockets (with different identifiers) attached to the same process.

If the client and server are using persistent HTTP, then throughout the duration of the persistent connection the client and server exchange HTTP messages via the same server socket. However, if the client and server use non-persistent HTTP, then a new TCP connection is created and closed for every request/response, and hence a new socket is created and later closed for every request/response. This frequent creating and closing of sockets can severely impact the performance of a busy web server (although a number of operating system tricks can be used to mitigate the problem).

1-4 Connection less transport: UDP

UDP like its cousin the Transmission Control Protocol (TCP) sits directly on top of the base Internet Protocol (IP). In general, UDP implements a fairly "lightweight" layer above the Internet Protocol. It seems at first site that similar service is provided by both UDP and IP, namely transfer of data. But we need UDP for multiplexing/demultiplexing of addresses.

UDP's main purpose is to abstract network traffic in the form of datagram’s. A datagram comprises one single "unit" of binary data; the first eight (8) bytes of a datagram contain the header information and the remaining bytes contain the data itself.

UDP Headers the UDP header consists of four (4) fields of two bytes each:

Source Port	Destination Port
length	checksum

- source port number
- destination port number
- datagram size
- checksum

UDP port numbers allow different applications to maintain their own "channels" for data; both UDP and TCP use this mechanism to support multiple applications sending and receiving data concurrently. The sending application (that could be a client or a server) sends UDP datagram's through the source port, and the recipient of the packet accepts this datagram through the destination port. Some applications use static port numbers that are reserved for or registered to the application. Other applications use dynamic (unregistered) port numbers. Because the UDP port headers are two bytes long, valid port numbers range from 0 to 65535; by convention, values above 49151 represent dynamic ports.

The datagram size is a simple count of the number of bytes contained in the header and data sections. Because the header length is a fixed size, this field essentially refers to the length of the variable-sized data portion (sometimes called the payload). The maximum size of a datagram varies depending on the operating environment. With a two-byte size field, the theoretical maximum size is 65535 bytes. However, some implementations of UDP restrict the datagram to a smaller number -- sometimes as low as 8192 bytes.

UDP checksums work as a safety feature. The checksum value represents an encoding of the datagram data that is calculated first by the sender and later by the receiver. Should an individual datagram be tampered with (due to a hacker) or get corrupted during transmission (due to line noise, for example), the calculations of the sender and receiver will not match, and the UDP protocol will detect this error. The algorithm is not fool-proof, but it is effective in many cases. In UDP, check summing is optional turning it off squeezes a little extra performance from the system as opposed to TCP where checksums are mandatory. It should be remembered that check summing is optional only for the sender, not the receiver. If the sender has used checksum then it is mandatory for the receiver to do so.

Usage of the Checksum in UDP is optional. In case the sender does not use it, it sets the checksum field to all 0's. Now if the sender computes the checksum then the recipient must also compute the checksum an set the field accordingly. If the checksum is calculated and turns out to be all 1's then the sender sends all 1's instead of all 0's. This is since in the algorithm for checksum computation used by UDP, a checksum of all 1's if equivalent to a checksum of all 0's. Now the checksum field is unambiguous for the recipient, if it is all 0's then checksum has not been used, in any other case the checksum has to be computed.

2. Principles of Reliable Data Transfer

Structure

Objectives

- Discuss reliable data transfer protocols
- Discuss about Pipelined Reliable Data Transfer Protocols
- Discuss sliding window protocols

2-1 Introduction

In this lesson, we consider the problem of reliable data transfer in a general context. This is appropriate since the problem of implementing reliable data transfer occurs not only at the transport layer, but also at the link layer and the application layer as well. The general problem is thus of central importance to networking. Indeed, if one had to identify a "top ten" list of fundamentally important problems in all of networking, this would be a top candidate to lead that list.lIn the next section we will examine TCP and show, in particular, that TCP exploits many of the principles that we are about to describe.

Figure 2-1 illustrates the framework for our study of reliable data transfer. The service abstraction provided to the upper-layer entities are that of a reliable channel through which data can be transferred. With a reliable channel, no transferred data bits are corrupted (flipped from 0 to 1, or vice versa) or lost, and all are delivered in the order in which they were sent. This is precisely the service model offered by TCP to the Internet applications that invoke it.

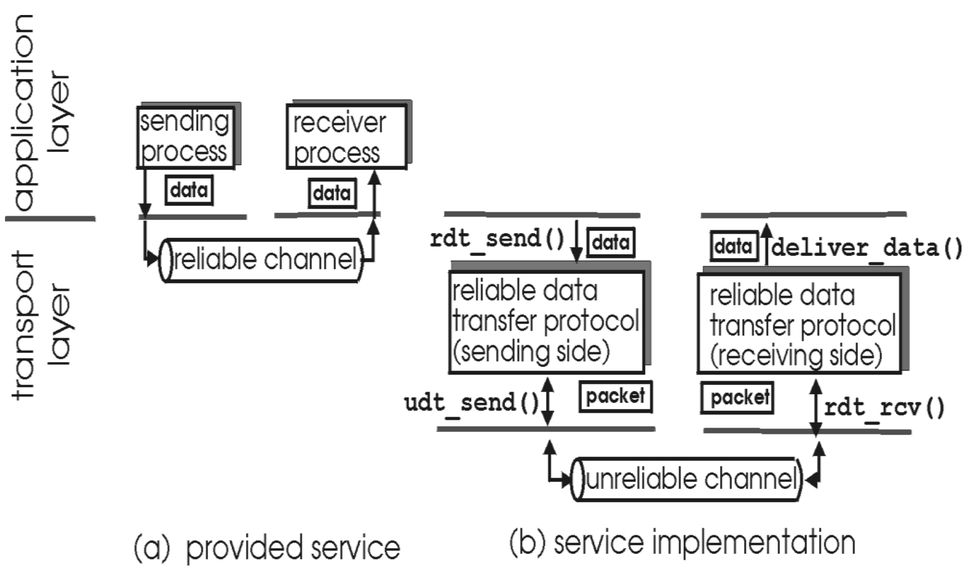


Figure2-1 Reliable data transfer: Service model and service implimentation

TCP is the responsibility of a **reliable data transfer protocol** to implement these service abstractions. This task is made difficult by the fact that the layer *below* the reliable data transfer protocol may be unreliable. For example, TCP is a reliable data transfer protocol that is implemented on top of an unreliable (IP) end-to-end network layer. More generally, the layer beneath the two reliably communicating endpoints might consist of a single physical link (as in the case of a link-level data transfer protocol) or a global internetwork (as in the case of a transport-level protocol). For our purposes, however, we can view this lower layer simply as an unreliable point-to-point channel.

In this lesson, we will incrementally develop the sender and receiver sides of a reliable data transfer protocol, considering increasingly complex models of the underlying channel. Figure 2-1(b) illustrates the interfaces for our data transfer protocol. The sending side of the data transfer protocol will be invoked from above by a call to rdt_send (). It will pass the data to be delivered to the upper layer at the receiving side. (Here rdt stands for "reliable data transfer" protocol and _send indicates that the sending side of rdt is being called. The first step in developing any protocol is to choose a good name!) On the receiving side, rdt_rcv( ) will be called when a packet arrives from the receiving side of the channel. When the rdt protocol wants to deliver data to the upper layer, it will do so by calling deliver_data (). In the following we use the terminology "packet" rather than "segment" for the protocol data unit. Because the theory developed in this section applies to computer networks in general and not just to the Internet transport layer, the generic term "packet" is perhaps more appropriate here.

In this lesson we consider only the case of **unidirectional** data transfer, that is, data transfer from the sending to the receiving side. The case of reliable **bidirectional** (that is, full-duplex) data transfer is conceptually no more difficult but considerably more tedious to explain. Although we consider only unidirectional data transfer, it is important to note that the sending and receiving sides of our protocol will nonetheless need to transmit packets in *both* directions, as indicated in Figure 2-1. We will see shortly that, in addition to exchanging packets containing the data to be transferred, the sending and receiving sides of rdt will also need to exchange control packets back and forth. Both the send and receive sides of rdt send packets to the other side by a call to udt send ( ) (where udt stands for "unreliable data transfer").

2-2 Building a Reliable Data Transfer Protocol

We now step through a series of protocols, each one becoming more complex, arriving at a flawless reliable data transfer protocol.

2-2-1 Reliable Data Transfer over a Perfectly Reliable Channel: rdt1.0

We first consider the simplest case, in which the underlying channel is completely reliable. The protocol itself, which we'll call rdt1.0, is trivial. The **finite-state** machine (FSM) definitions for the rdt 1.0 sender and receiver are shown in Figure 2-2. The FSM in Figure 2-2(a) defines the operation of the sender, while the FSM in Figure 2-2(b) defines the operation of the receiver. It is important to note that there is a *separate* FSM for the sender and for the receiver. The sender and receiver FSMs in Figure 2-2 each have just one state. The arrows in the FSM description indicate the transition of the protocol from one state to another. (Since each FSM in Figure 2-2 has just one state, a transition is necessarily from the one state back to itself; we'll see more complicated state diagrams shortly.) The event causing the transition is shown above the horizontal line labeling the transition, and the actions taken when the event occurs are shown below the horizontal line When no action is taken on an event, or no event occurs and an action is taken, we'll use the symbol Λ below or above the horizontal respectively to explicitly denote the lack of an action or event. The initial state of the FSM is indicated by the dashed arrow. Although the FSMs in Figure 2-2 have but one state, the FSMs we will see shortly have multiple states, so it will be important to identify the initial state of each FSM.

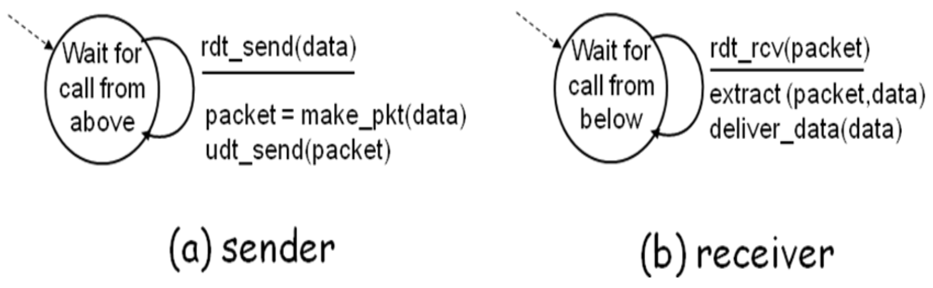


Figure 2-1 rdt1.0 A protocol for a completely reliable channel

The sending side of rdt simply accepts data from the upper layer via the rdt_send(data) event, creates a packet containing the data (via the action make_pkt(data)) and sends the packet into the channel. In practice, the rdt send (data) event would result from a procedure call (for example, to rdt_send ()) by the upper-layer application.

On the receiving side, rdt receives a packet from the underlying channel via the rdt_rev (packet) event, removes the data from the packet (via the action extract (packet, data)) and passes the data up to the upper layer (via the action deliver_data(data)). In practice, the rdt_rev (packet) event would result from a procedure call (for example, to rdt_rcv ( )) from the lowear layer protocol.

In this simple protocol, there is no difference between a unit of data and a packet. Also, all packet flow is from the sender to receiver; with a perfectly reliable channel there is no need for the receiver side to provide any feedback to the sender since nothing can go wrong! Note that we have also assumed that the receiver is able to receive data as fast as the sender happens to send data. Thus, there is no need for the receiver to ask the sender to "slow down!"

2-2-2 Reliable Data Transfer over a Channel with Bit Errors: rdt2. 0

A more realistic model of the underlying channel is one in which bits in a packet may be corrupted. Such bit errors typically occur in the physical components of a network as a packet is transmitted, propagates, or is buffered. We'll continue to assume for the moment that all transmitted packets are received (although their bits may be corrupted) in the order in which they were sent.

Before developing a protocol for reliably communicating over such a channel, first consider how people might deal with such a situation. Consider how you yourself might dictate a long message over the phone. In a typical scenario, the message taker might say "OK" after each sentence has been heard, understood, and recorded. If the message taker hears a garbled sentence, you're asked to repeat the garbled sentence. This message-dictation protocol uses both **positive acknowledgments** ("OK") and **negative acknowledgments** ("Please repeat that."). These control messages allow the receiver to let the sender know what has been received correctly, and what has been received in error and thus requires repeating. In a computer network setting, reliable data transfer protocols based on such retransmission are known as **ARQ** (Automatic Repeat reQuest) **protocols.**

Fundamentally, three additional protocol capabilities are required in ARQ protocols to handle the presence of bit errors:

- *Error detection.* First, a mechanism is needed to allow the receiver to detect when bit errors have occurred. Recall from the previous lesson that UDP uses the Internet checksum field for exactly this purpose. In further lessons we'll examine error-detection and -correction techniques in greater detail; these techniques allow the receiver to detect and possibly correct packet bit errors. For now, we need only know that these techniques require that extra bits (beyond the bits of original data to be transferred) be sent from the sender to the receiver; these bits will be gathered into the packet checksum field of the rdt2 .0 data packet.

- *Receiver feedback.* Since the sender and receiver are typically executing on different end systems, possibly separated by thousands of miles, the only way for the sender to learn of the receiver's view of the world (in this case, whether or not a packet was received correctly) is for the receiver to provide explicit feedback to the sender. The positive (ACK) and negative (NAK) acknowledgment replies in the message-dictation scenario are examples of such feedback. Our rdt2.0 protocol will similarly send ACK and NAK packets back from the receiver to the sender. In principle, these packets need only be one bit long; for example, a 0 value could indicate a NAK and a value of 1 could indicate an ACK.

- *Retransmission. A* packet that is received in error at the receiver will be retransmitted by the sender.

Figure 2-3 shows the FSM representation of rdt2.0, a data transfer protocol employing error detection, positive acknowledgments, and negative acknowledgments.

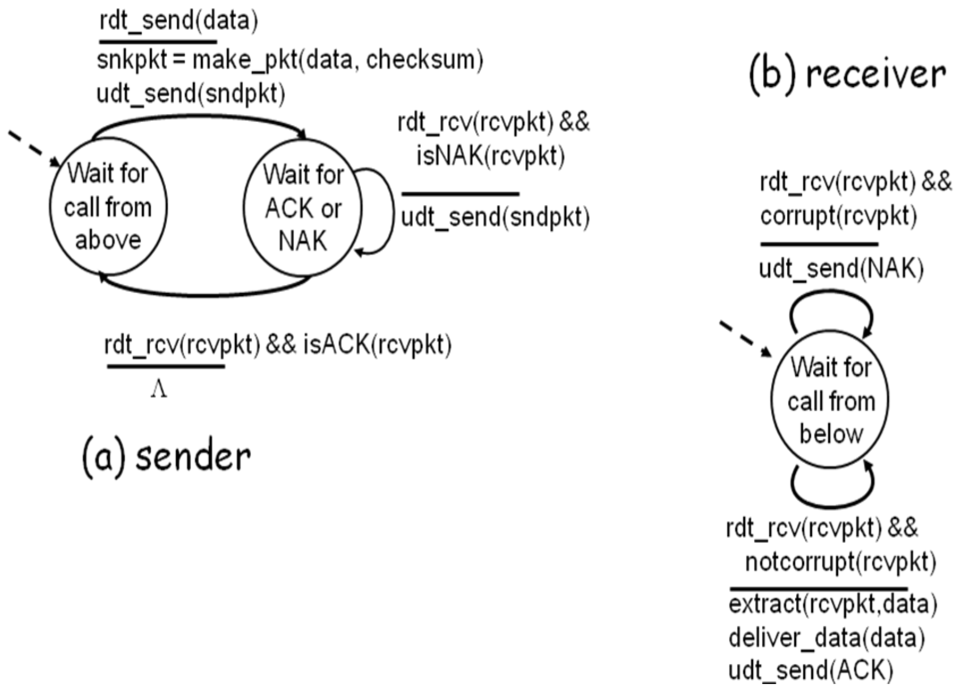


Figure 2-3 rdt2.0 a protocol for a channel with bit errors

The send side of rdt2.0 has two states. In the leftmost state, the send-side protocol is waiting for data to be passed down from the upper layer. When the rdt_send (data) event occurs, the sender will create a packet (sndpkt) containing the data to be sent, along with a packet checksum, and then send the packet via the udt_send (sndpkt) operation. In the rightmost state, the sender protocol is waiting for an ACK or a NAK packet from the receiver. If an ACK packets received (the notation rdt_rcv (rcvpkt) & & isACK (rcvpkt) in Figure 2-3 corresponds to this event), the sender knows that the most recently transmitted packet has been received correctly and thus the protocol returns to the state of waiting for data from the upper layer. If a NAK is received, the protocol retransmits the last packet and waits for an ACK or NAK to be returned by the receiver in response to the retransmitted data packet. It is important to note that when the receiver is in the wait-for-ACK-or-NAK state, it *cannot* get more data from the upper layer; that will happen only after the sender receives an ACK and leaves this state. Thus, the sender will not send a new piece of data until it is sure that the receiver has correctly received the current packet. Because of this behavior, protocols such as rdt2.0 are known as **stop-and-wait** protocols.

The receiver-side FSM for rdt2.0 still has a single state. On packet arrival the receiver replies with either an ACK or a NAK, depending on whether or not the received packet is corrupted. In Figure 2-3, the notation rdt_rcv(rcvpkt) && corrupt (rcvpkt) corresponds to the event in which a packet is received and is found to be in error.

Protocol rdt2.0 may look as if it works but, unfortunately, it has a fatal flaw. In particular, we haven't accounted for the possibility that the ACK or NAK packet could be corrupted! (Before proceeding on, you should think about how this problem may be fixed.) Unfortunately, our slight oversight is not as innocuous as it may seem. Minimally, we will need to add checksum bits to ACK/NAK packets in order to detect such errors. The more difficult question is how the protocol should recover from errors in ACK or NAK packets. The difficulty here is that if an ACK or NAK is corrupted, the sender has no way of knowing whether or not the receiver has correctly received the last piece of transmitted data.

Consider three possibilities for handling corrupted ACKs or NAKs:

- For the first possibility, consider what a human might do in the message-dictation scenario. If the speaker didn't understand the "OK" or "Please repeat that" reply from the receiver, the speaker would probably ask "What did you say?" (thus introducing a new type of sender-to-receiver packet to our protocol). The speaker would then repeat the reply. But what if the speaker's "What did you say?" is corrupted? The receiver, having no idea whether the garbled sentence was part of the dictation or a request to repeat the last reply, would probably then respond with "What did *you* say?" And then, of course, that response might be garbled. Clearly, we're heading down a difficult path.

- A second alternative is to add enough checksum bits to allow the sender not only to detect, but to recover from, bit errors. This solves the immediate problem for a channel that can corrupt packets but not lose them.

- A third approach is for the sender simply to resend the current data packet when it receives a garbled ACK or NAK packet. This approach, however, introduces **duplicate packets** into the sender-to-receiver channel. The fundamental difficulty with duplicate packets is that the receiver doesn't know whether the ACK or NAK it last sent was received correctly at the sender. Thus, it cannot know a priori whether an arriving packet contains new data or is a retransmission!

A simple solution to this new problem (and one adopted in almost all existing data transfer protocols, including TCP) is to add a new field to the data packet and have the sender number its data packets by putting a **sequence number** into this field. The receiver then need only check this sequence number to determine or not the received packet is a retransmission. For this simple case of a stop-and-wait protocol, a one-bit sequence number will suffice, since it will allow the receiver to know whether the sender is resending the previously transmitted packet (the sequence number of the received packet has the same sequence number as the most recently received packet) or a new packet (the sequence number changes, moving "forward" in modulo-2 arithmetic). Since we are currently assuming a channel that does not lose packets, ACK and NAK packets do not themselves need to indicate the sequence number of the packet they are acknowledging. The sender knows that a received ACK or NAK packet (whether garbled or not) was generated in response its most recently transmitted data packet.

Figures 2-4 and 2-5 show the FSM description for rdt2.1, our fixed version of rdt2.0. The rdt2.1 sender and receiver FSMs each now have twice as many states as before. This is because the protocol state must now reflect whether the packet currently being sent (by the sender) or expected (at the receiver) should have a sequence number of 0 or 1. Note that the actions in those states where a 0-numbered packet is being sent or expected are mirror images of those where a 1-numbered packet is being sent or expected; the only differences have to do with the handling of the sequence number.

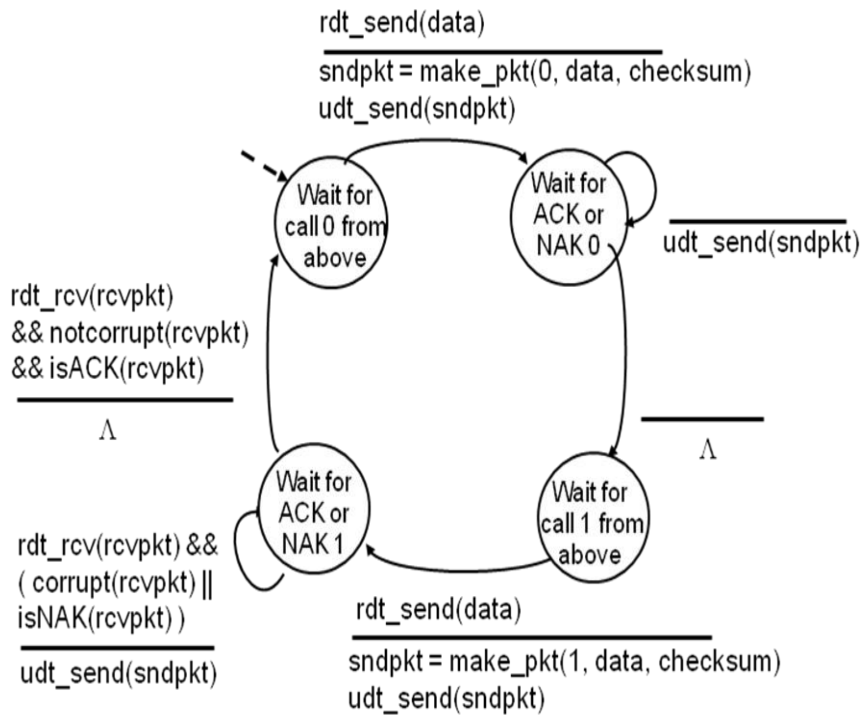

rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
Wait for call 0 from above
Wait for ACK or NAK 0
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt)
Λ
Λ
Wait for ACK or NAK 1
Wait for call 1 from above
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isNAK(rcvpkt))
udt_send(sndpkt)
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)

Figure2-4 rdt2.1 sender

Protocol rdt2.1 uses both positive and negative acknowledgments from the receiver to the sender. When an out-of-order packet is received, the receiver sends a positive acknowledgment for the packet it has received. When a corrupted packet is received, the receiver sends a negative acknowledgment. We can accomplish the same effect as a NAK if, instead of sending a NAK, we instead send an ACK for the last correctly received packet. A sender that receives two ACKs for the same packet (that is, receives **duplicate ACKs)** knows that the receiver did not correctly receive the packet following the packet that is being ACKed twice. Our NAK-free reliable data transfer protocol for a channel with bit errors is rdt2 .2, shown in Figures 2-6. One subtle change between rtdt2.1 and rdt2.2 is that the receiver must now include the sequence number of the packet being acknowledged by an ACK message (this is done by including the ACK.,0 or ACK.,1 argument in make_pkt ( ) in the receiver FSM), and the sender must now check the sequence number of the packet being acknowledged by a received ACK message (this is done by including the 0 or 1 argument in isACK () in the sender FSM).

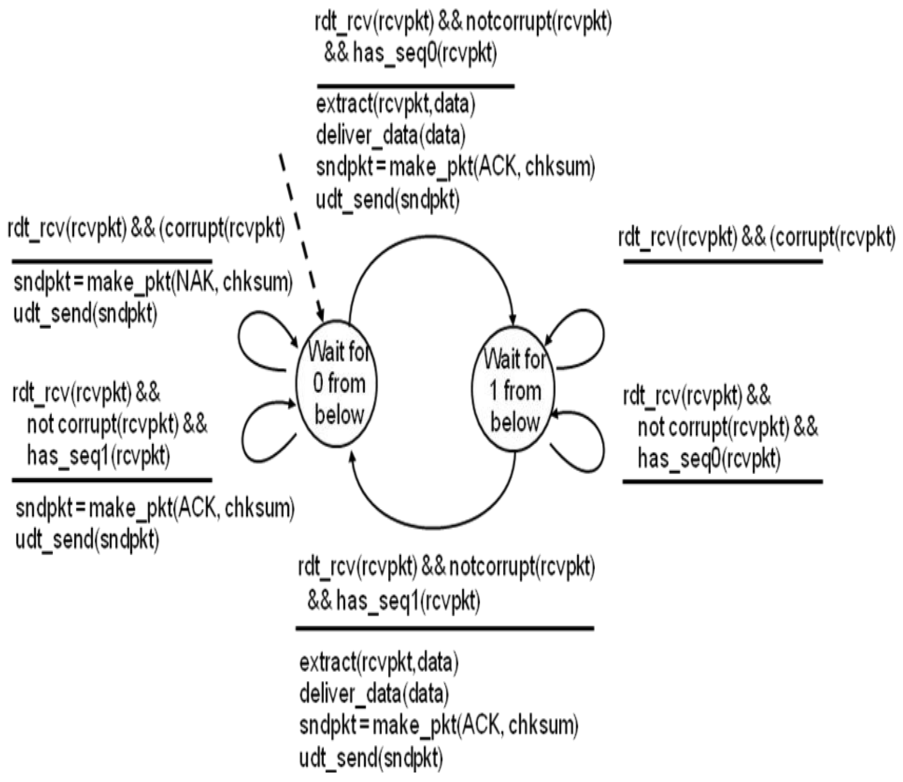

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq0(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)
sndpkt=make_pkt(NAK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)
Wait for 0 from below
Wait for 1 from below
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && has_seq1(rcvpkt)
sndpkt=make_pkt(ACK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && has_seq0(rcvpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK, chksum)
udt_send(sndpkt)

Figure 2-5 rdt2-1 receiver

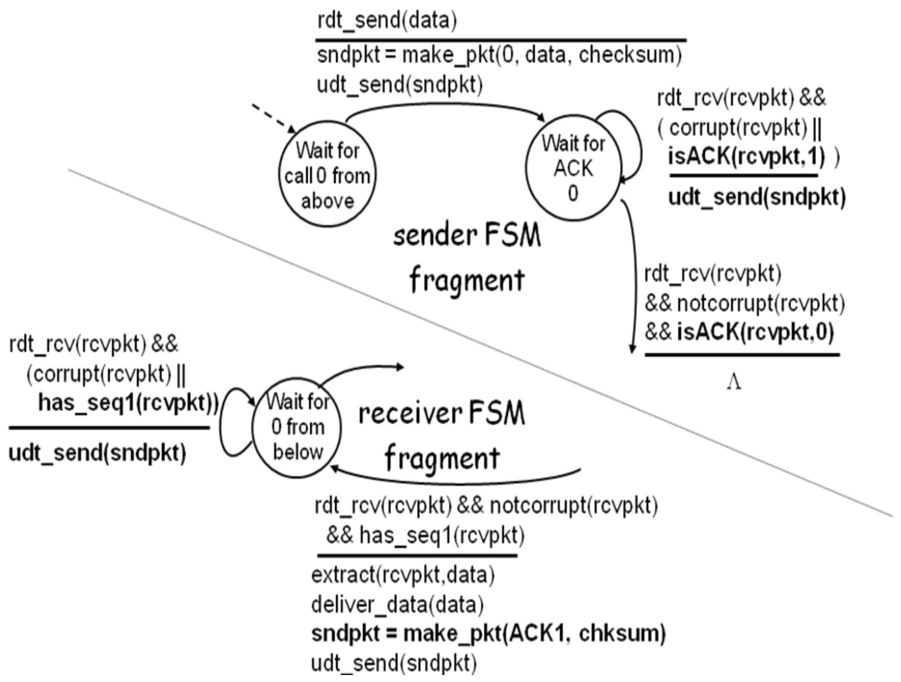

rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
Wait for call 0 from above
Wait for ACK 0
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
udt_send(sndpkt)
sender FSM fragment
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
Λ
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || has_seq1(rcvpkt))
udt_send(sndpkt)
Wait for 0 from below
receiver FSM fragment
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK1, chksum)
udt_send(sndpkt)

Figure 2-6 rdt2.2 sender

2-2-3 Reliable Data Transfer over a Lossy Channel with Bit Errors: rdt3.0

Suppose now that in addition to corrupting bits, the underlying channel can lose packets as well, a not-uncommon event in today's computer networks (including the Internet). Two additional concerns must now be addressed by the protocol: how to detect packet loss and what to do when packet loss occurs. The use of check summing, sequence numbers, ACK packets, and retransmissions the techniques ready developed in rdt2.2 will allow us to answer the latter concern. Handling the first concern will require adding a new protocol mechanism.

There are many possible approaches toward dealing with packet. Here, we'll put the burden of detecting and recovering from lost packets on the sender. Suppose that the sender transmits a data packet and either that packet, or the receiver's ACK of that packet, gets lost. In either case, no reply is forthcoming at the sender from the receiver. If the sender is willing to wait long enough so that it is certain that packet has been lost, it can simply retransmit the data packet. You should convince yourself that this protocol does indeed work.

But how long must the sender wait to be certain that something has been lost? The sender must clearly wait at least as long as a round-trip delay between the sender and receiver (which may include buffering at intermediate routers) plus what ever amount of time is needed to process a packet at the receiver. In many networks, this worst-case maximum delay is very difficult even to estimate, much less know with certainty. Moreover, the protocol should ideally recover from packet loss as soon as possible; waiting for a worst-case delay could mean a long wait until error recovery is initiated. The approach thus adopted in practice is for the sender to "ju-diciously" choose a time value such that packet loss is likely, although not guaranteed, to have happened. If an ACK is not received within this time, the packet is .retransmitted. Note that if a packet experiences a particularly large delay, the sender may retransmit the packet even though neither the data packet nor its ACK have been lost. This introduces the possibility of **duplicate data packets** in the sender-to-receiver channel. Happily, protocol rdt2.2 already has enough functionality (that is, sequence numbers) to handle the case of duplicate packets.

From the sender's viewpoint, retransmission is a panacea. The sender does not know whether a data packet was lost, an ACK was lost, or if the packet or ACK was simply overly delayed. In all cases, the action is the same: retransmit. In order to implement a time-based retransmission mechanism, a **countdown timer** will be needed that can interrupt the sender after a given amount of time has expired. The sender will thus need to be able to (1) start the timer each time a packet (either a first-time packet, or a retransmission) is sent, (2) respond to a timer interrupt (taking) appropriate actions), and (3) stop the timer.

The existence of sender-generated duplicate packets and packet (data, ACK) loss also complicates the sender's processing of any ACK packet it receives. If I ACK is received, how is the sender to know if it was sent by the receiver in response to its (sender's) own most recently transmitted packet, or is a delayed ACK sent response to an earlier transmission of a different data packet? The solution to this dilemma is to augment the ACK packet with an **acknowledgment Held.** When the receiver generates an ACK, it will copy the sequence number of the data packet being acknowledged into this acknowledgment field. By examining the contents of the acknowledgment field, the sender can determine the sequence number of the packet being positively acknowledged.

Figure 2-7 shows the sender FSM for rdt3.0, a protocol that reliably transfers data over a channel that can corrupt or lose packets. Figure 2-8 shows how the protocol operates with no lost or delayed packets and how it handles lost data packets. In Figure 2-8, time moves forward from the top of the diagram toward the bottom of the diagram; note that a receive time for a packet is necessarily later than the send time for a packet as a result of transmission and propagation delays. In Figures 2-8b-d, the send-side brackets indicate the times at which a timer is set and later times out. Because packet sequence numbers alternate between 0 and 1, protocol rdt3.0 is sometimes known as the **alternating-bit protocol.**

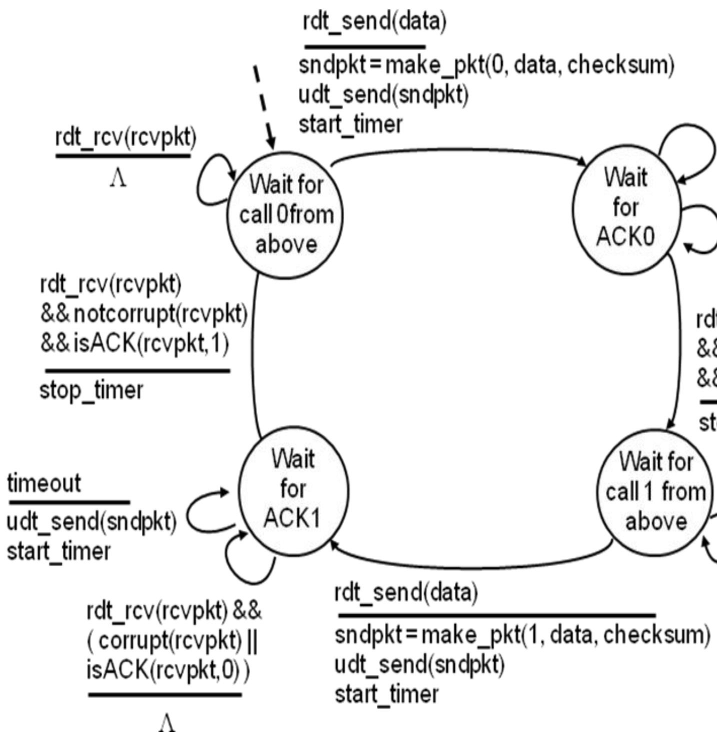
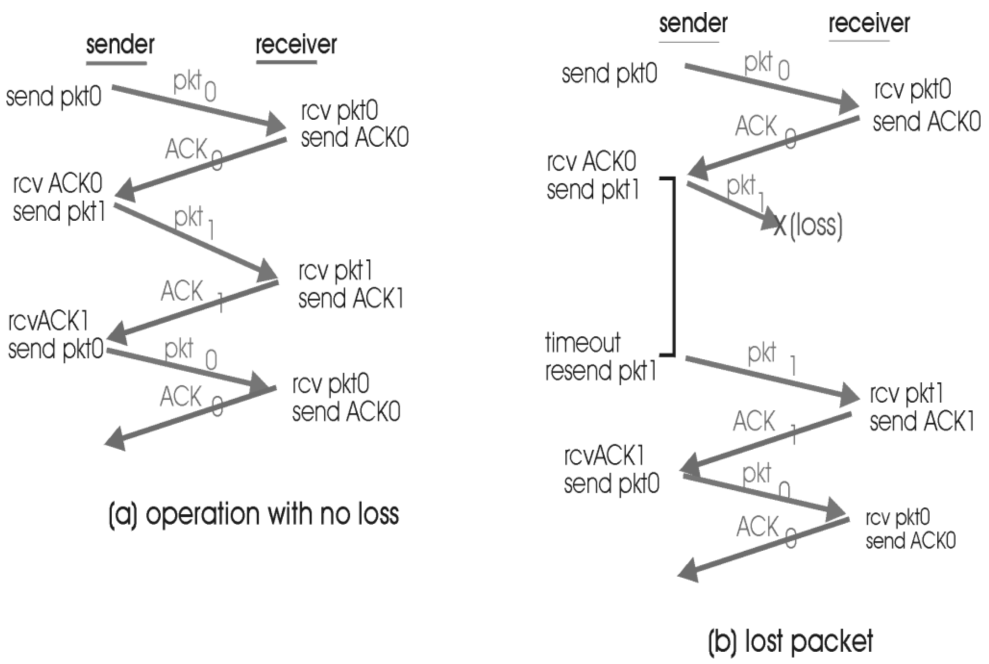

rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt)
Λ
Wait for call 0from above
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
Λ
Wait for ACK0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1)
stop_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
stop_timer
timeout
udt_send(sndpkt)
start_timer
Wait for ACK1
Wait for call 1 from above
rdt_rcv(rcvpkt)
Λ
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,0))
Λ
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)
start_timer

Figure 2-7 rdt3.0 sender



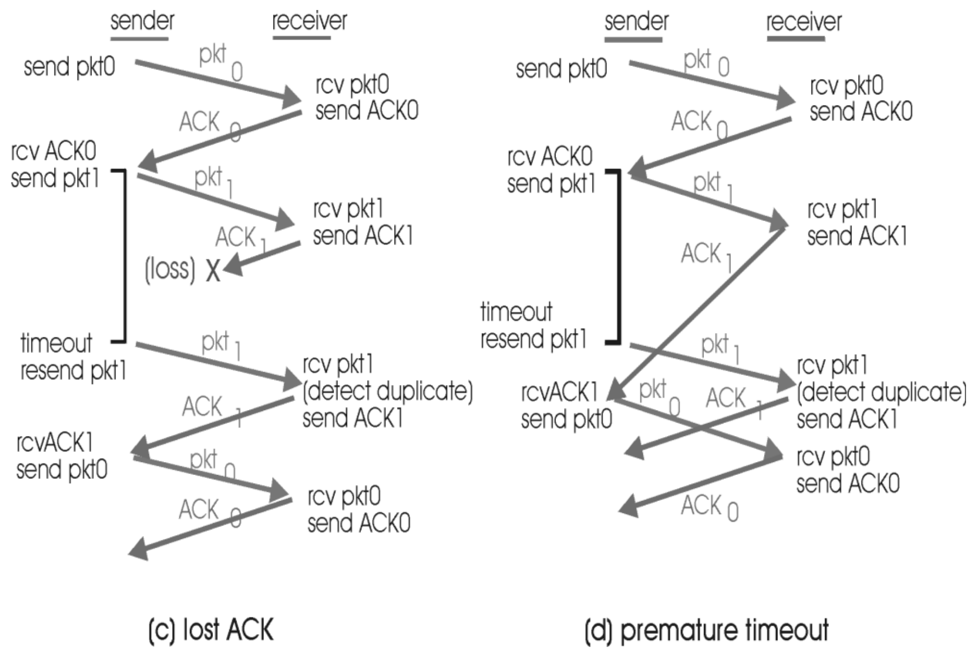


Figure 2-8 operation of rdt3.0, the alternating-bit protocol

We have now assembled the key elements of a data transfer protocol. Checksums, sequence numbers, timers, and positive and negative acknowledgment packets each play a crucial and necessary role in the operation of the protocol. We now have a working reliable data transfer protocol!

2-3 Pipelined Reliable Data Transfer Protocols

Protocol rdt3.0 is a functionally correct protocol, but it is unlikely that anyone would be happy with its performance, particularly in today's high-speed networks. At the heart of rdt3.0's performance problem is the fact that it is a stop-and-wait protocol.

To appreciate the performance impact of this stop-and-wait behavior, consider an idealized case of two end hosts, one located on the West Coast of the United States and the other located on the East Coast, as shown in Figure 2-9. The speed of-light round-trip propagation delay between these two end systems, RTT, is approximately 30 milliseconds. Suppose that they are connected by a channel with a transmitted rate, *R,* of 1Gbps (10^9 bits per second). With a packet size, *L,* of 1000 bytes (8000 bits) per packet, including" both header fields and data, the time needed to actually transmit the packet into the 1Gbps link is:

$$t_{transmit} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8\text{kb/pkt}}{10^9 \text{ b/sec}} = 8 \text{ microsec}$$

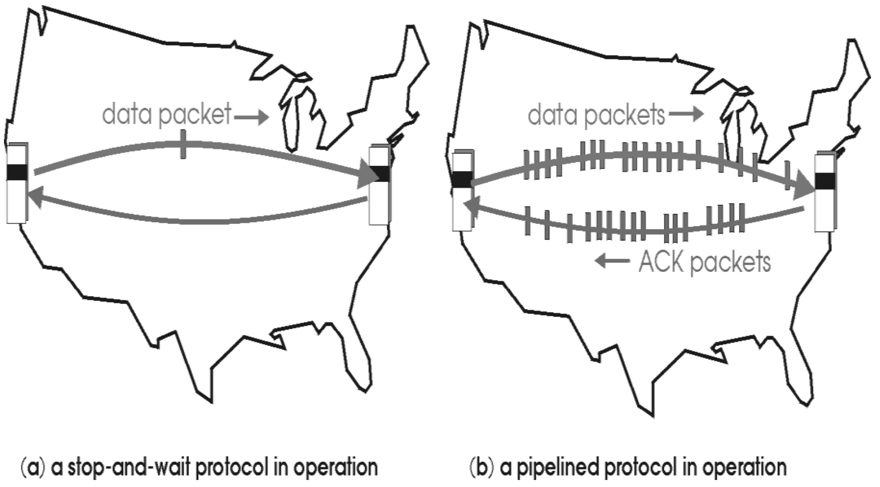

data packet
data packets
ACK packets
(a) a stop-and-wait protocol in operation
(b) a pipelined protocol in operation

Figure 2-9 Stop-and wait versus pipelined protocol

Figure 2-10(a) shows that with our stop-and-wait protocol, if the sender begins sending the packet at *t = 0,* then at *t* = *L/R* = 8 microseconds, the last bit enters the channel at the sender side. The packet then makes its 15-msec cross-country journey, with the last bit of the packet emerging at the receiver at *t* = RTT/2 + *L/R* = 15.008 msec. Assuming for simplicity that ACK packets are extremely small (so that we can ignore their transmission time) and that the receiver can send an ACK as soon as the last bit of a data packet is received, the ACK emerges back at the sender at *t* = RTT + *L/R* = 30.008 msec. At this point, the sender can now transmit the next message. Thus, in 30.008 msec, the sender was sending for only 0.008 msec. If we define the **utilization** of the sender (or the channel) as the fraction of time the sender is actually busy sending bits into the channel, the analysis in Figure 2-10(a) shows that the stop-and-wait protocol has rather dismal sender utilization, U_{sender}, of

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

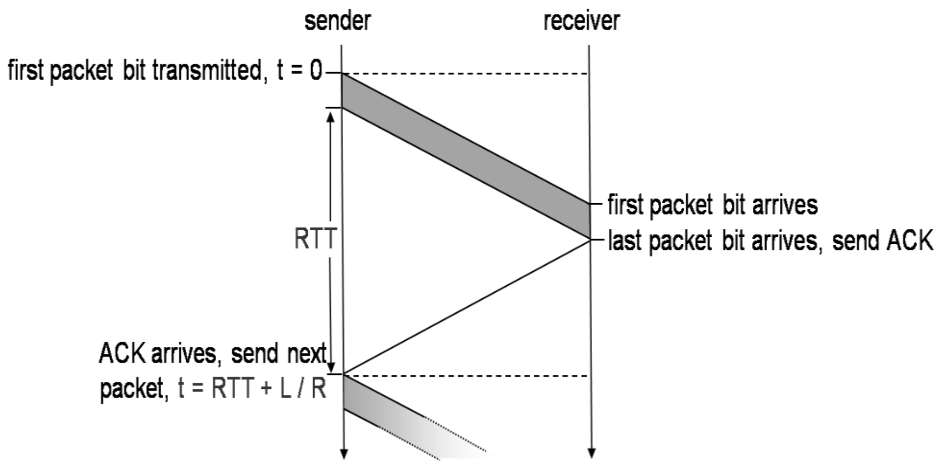
That is, the sender was busy only 2.7 hundredths of one percent of the time. Viewed another way, the sender was able to send only 1000 bytes in 30.008 milliseconds, an effective throughput of only 267 kbps even though a 1Gbps link was available! Imagine the unhappy network manager who just paid a fortune for a gigabit capacity link but manages to get a throughput of only 267 kilobits per second! This is a graphic example of how network protocols can limit the capabilities provided by the underlying network hardware. Also, we have neglected lower-layer protocol-processing times at the sender and receiver, as well as the processing and queuing delays that would occur at any intermediate routers between the sender and receiver. Including these effects would serve only to further increase the delay and further accentuate the poor performance.

The solution to this particular performance problem is a simple one: rather than operate in a stop-and-wait manner, the sender is allowed to send multiple packets without waiting for acknowledgments, as illustrated in Figure 3.18(b) shows that if the sender is allowed to transmit three packets before having to wait for acknowledgments, the utilization of the sender is essentially tripled. Since the many in-transit sender-to-receiver packets can be visualized as filling a pipeline, this technique is known as **pipelining.** Pipelining has the following consequences for reliable data transfer protocols:

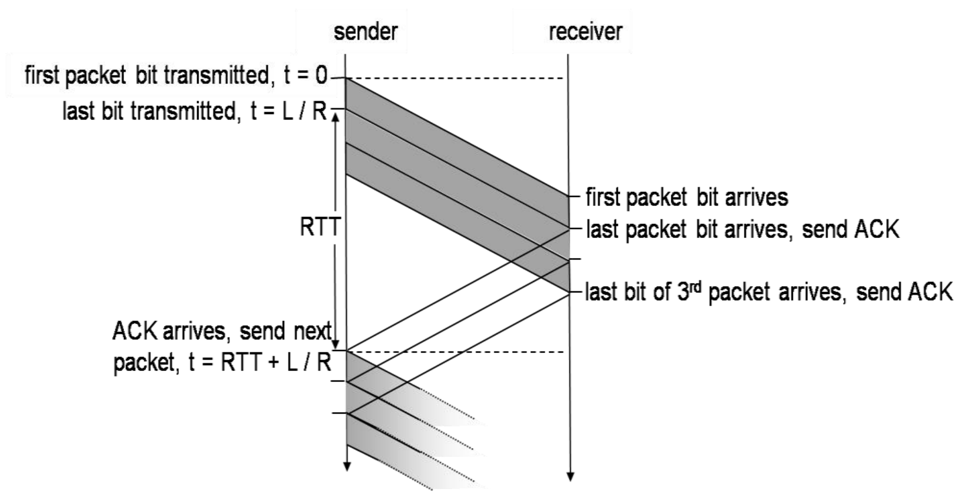
• The range of sequence numbers must be increased, since each in-transit packet (not counting retransmissions) must have a unique sequence number and there may be multiple, in-transit, unacknowledged packets.

• The sender and receiver sides of the protocols may have to buffer more than one packet. Minimally, the sender will have to buffer packets that have been transmitted but not yet acknowledged. Buffering of correctly received packets may also be needed at the receiver, as discussed below.

The range of sequence numbers needed and the buffering requirements will depend on the manner in which a data transfer protocol responds to lost, corrupted, and overly delayed packets. Two basic approaches toward pipelined error recovery can be identified: **Go-Back-N** and **selective repeat.**


sender
receiver
first packet bit transmitted, t = 0
first packet bit arrives
last packet bit arrives, send ACK
RTT
ACK arrives, send next packet, t = RTT + L / R

(a) stop-and-wait operation


sender
receiver
first packet bit transmitted, t = 0
last bit transmitted, t = L / R
RTT
ACK arrives, send next packet, t = RTT + L / R
first packet bit arrives
last packet bit arrives, send ACK
last bit of 3rd packet arrives, send ACK

(b) Pipeline operation

Figure 2-10 Stop-and-wait and pipelined sending

2-4 Go-Back-N (GBN)

In a Go-Back-N (GBN) protocol, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, *N, of* unacknowledged packets in the pipeline. Figure 2-11 shows the sender's view of the range of sequence numbers in a GBN protocol. We define base to be the sequence number of the oldest unacknowledged packet and nextseqnum to be the smallest unused sequence number (that is, the sequence number of the next packet to be sent), then four intervals in the range of sequence numbers can be identified. Sequence numbers in the interval [0, base-1] correspond to packets that have already been transmitted and acknowledged. The interval [base, nextseqnum-1] corresponds to packets that have been sent but not yet acknowledged. Sequence numbers in the interval [nextseqnum, base+N-1] can be used for packets that can be sent immediately, should data arrive from the upper layer. Finally, sequence numbers greater than or equal to base+N cannot be used until an unacknowledged packet currently in the pipeline (specifically, the packet with sequence number base) has been acknowledged.

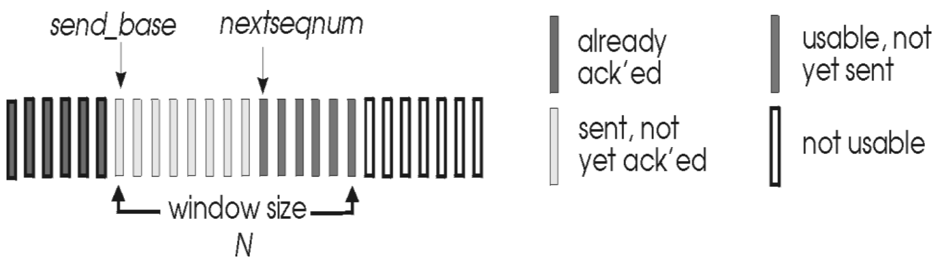


Figure2-11 sender's view of sequence numbers in go-back-N

As suggested by Figure 2-11, the range of permissible sequence numbers for transmitted but not yet acknowledged packets can be viewed as a "window" of size iV6ver the range of sequence numbers. As the protocol operates this window slides forward over the sequence number space. For this reason, *N* is often referred to as the **window size** and the GBN protocol itself as a **sliding-window protocol.** You might be wondering why we would even limit the number of outstanding, unacknowledged packets to a value of *N* in the first place.

In practice, a packet's sequence number is carried in a fixed-length field in the packet header. If *k is* the number of bits in the packet sequence number field, the range of sequence numbers is thus $[0,2^k - 1]$. With a finite range of sequence numbers, all arithmetic involving sequence numbers must then be done using modulo 2^K arithmetic. (That is, the sequence number space can be thought of as a ring of size 2^k , where sequence number $2^k -1$ is immediately followed by sequence number 0.) Recall that rdt3.0 had a one-bit sequence number and a range of sequence numbers of [0,1]. Several of the problems at the end of this chapter explore the consequences of a finite range of sequence numbers.

Figures 2-12 and 2-13 give an extended FSM description of the sender and receiver sides of an ACK-based, NAK-free, GBN protocol. We refer to this FSM description as an extended FSM since we have added variables (similar to programming-language variables) for base and nextseqnum and also added operations on these variables and conditional actions involving these variables. Note that the extended FSM specification is now beginning to look somewhat like a programming-language specification.

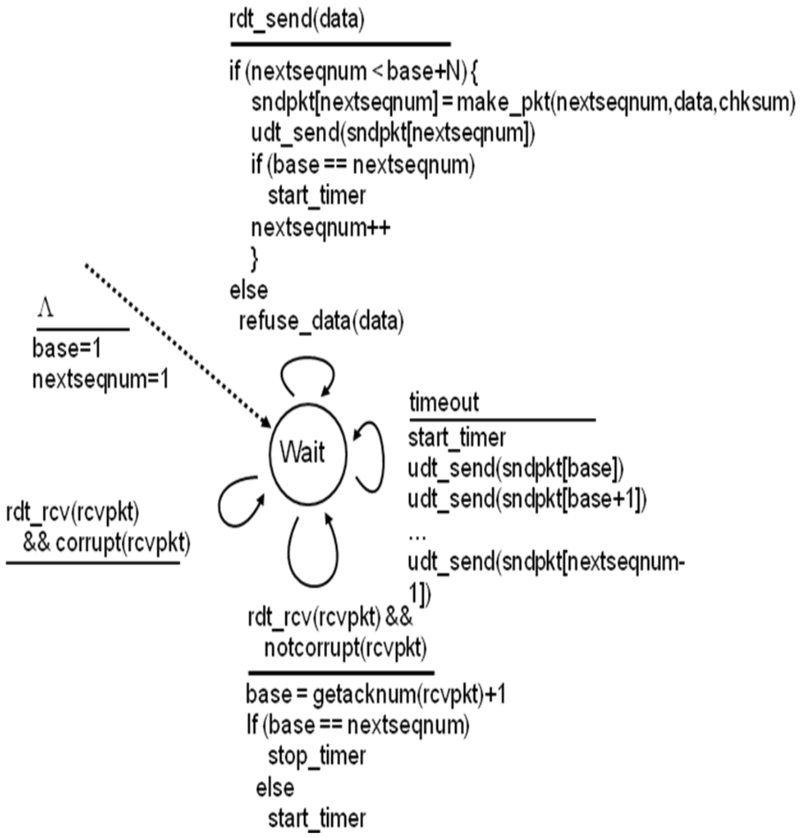

rdt_send(data)
if (nextseqnum < base+N) {
sndpkt[nextseqnum] = make_pkt(nextseqnum,data,chksum)
udt_send(sndpkt[nextseqnum])
if (base == nextseqnum)
start_timer
nextseqnum++
}
else
refuse_data(data)
Λ
base=1
nextseqnum=1
Wait
timeout
start_timer
udt_send(sndpkt[base])
udt_send(sndpkt[base+1])
...
udt_send(sndpkt[nextseqnum-1])
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
base = getacknum(rcvpkt)+1
If (base == nextseqnum)
stop_timer
else
start_timer

Figure2-12 Extended FSM description of GBN sender

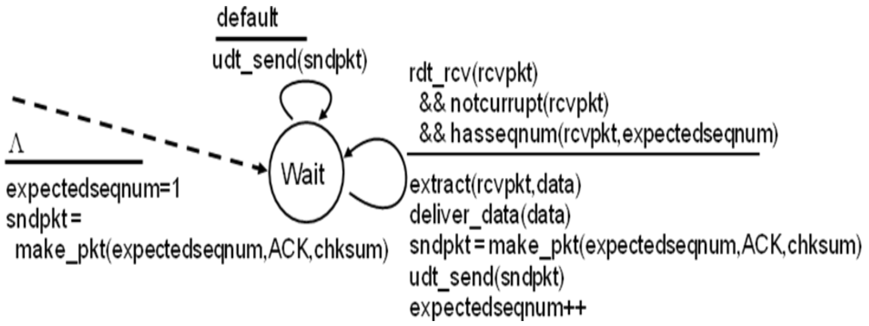


Figure2-13 Extended FSM description of GBN receiver

The GBN sender must respond to three types of events:

• *Invocation from above.* When rdt_send() is called from above, the sender first checks to see if the window is full, that is, whether there are *N* outstanding, unacknowledged packets. If the window is not full, a packet is created and sent, and variables are appropriately updated. If the window is full, the sender simply returns the data back to the upper layer, an implicit indication that the window is full. The upper layer would presumably then have to try again later. In a real implementation, the sender would more likely have either buffered (but not immediately sent) this data, or would have a synchronization mechanism (for example, a semaphore or a flag) that would allow the upper layer to call rdt_send() only when the window is not full.

• *Receipt of an ACK.* In our GBN protocol, an acknowledgment for packet with sequence number *n* will be taken to be a **cumulative acknowledgment,** indicating that all packets with a sequence number up to and including *n* have been correctly received at the receiver. We'll come back to this issue shortly when we examine the receiver side of GBN.

• *A timeout event.* The protocol's name, "Go-Back-N," is derived from the sender's behavior in the presence of lost or overly delayed packets. As in the stop-and-wait protocol, a timer will again be used to recover from lost data or acknowledgment packets. If a timeout occurs, the sender resends *all* packets that have been previously sent but that have not yet been acknowledged. Our sender in Figure 3.20 uses only a single timer, which can be thought of as a timer for the oldest transmitted but not yet acknowledged packet. If an ACK is received but there are still additional transmitted but not yet acknowledged packets, the timer is restarted. If there are no outstanding unacknowledged packets, the timer is stopped.

The receiver's actions in GBN are also simple. If a packet with sequence number n is received correctly and is in order (that is, the data last delivered to the upper layer came from a packet with sequence number *n* - 1), the receiver sends an ACK for packet *n* and delivers the data portion of the packet to the upper layer. In all other cases, the receiver discards the packet and resends an ACK for the most recently received in-order packet. Note that since packets are delivered one at a time to the upper layer, if packet *k* has been received and delivered, then all packets with a sequence number lower than *k* have also been delivered. Thus, the use of cumulative acknowledgments is a natural choice for GBN.

In our GBN protocol, the receiver discards out-of-order packets. Although it may seem silly and wasteful to discard a correctly received (but out-of-order) packet, there is some justification for doing so. Recall that the receiver must deliver data in order to the upper layer. Suppose now that packet *n* is expected, but packet *n* + 1 arrives. Since data must be delivered in order, the receiver *could* buffer (save) packet *n* + 1 and then deliver this packet to the upper layer after it had later received and delivered packet *n.* However, if packet n is lost, both it and packet *n* + 1 will eventually be retransmitted as a result of the GBN retransmission rule at the sender. Thus, the receiver can simply discard packet n + 1. The advantage of this approach is the simplicity of receiver buffering—the receiver need not buffer *any* out-of-order packets. Thus, while the sender must maintain the upper and lower bounds of its window and the position of nextseqnum within this window, the only piece of information the receiver need maintain is the sequence number of the next in-order packet. This value is held in the variable expectedseqnum, shown in the receiver FSM in Figure 2-13. Of course, the disadvantage of throwing away a correctly re-ceived packet is that the subsequent retransmission of that packet might be lost or garbled and thus even more retransmissions would be required.

Figure 2-14 shows the operation of the GBN protocol for the case of a window size of four packets. Because of this window size limitation, the sender sends packets 0 through 3 but then must wait for one or more of these packets to be acknowledged before proceeding. As each successive ACK (for example, ACKO and ACK1) is received, the window slides forward and the sender can transmit one new packet (pkt4 and pkt5, respectively). On the receiver side, packet 2 is lost and thus packets 3, 4, and 5 are found to be out of order and are discarded.

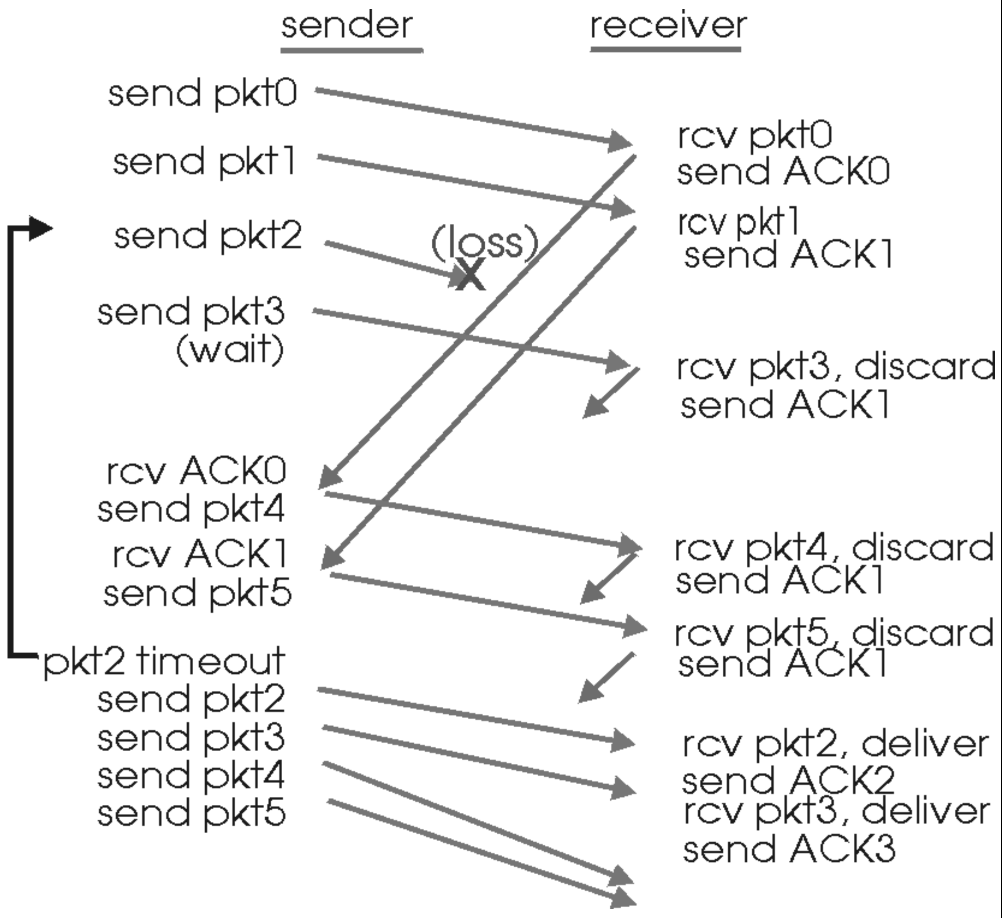


Figure2-14 Go-Back-N in operation

Before closing our discussion of GBN, it is worth noting that an implementation of this protocol in a protocol stack would likely have a structure similar to that of the extended FSM in Figure 2-12. The implementation would also likely be in the form of various procedures that implement the actions to be taken in response to the various events that can occur. In such **event-based programming,** the various procedures are called (invoked) either by .other procedures in the protocol stack, or as the result of an interrupt. In the sender, these events would be (1) a call from the upper-layer entity to invoke rdt_send ( ), (2) a timer interrupt, and (3) a call from the lower layer to invoke rdt_rev () when a packet arrives. The programming exercises at the end of this chapter will give you a chance to actually implement these routines in a simulated, but realistic, network setting.

2-5 Selective Repeat (SR)

The GBN protocol allows the sender to potentially "fill the pipeline" in Figure 2.9 with packets, thus avoiding the channel utilization problems we noted with stop-and-wait protocols. There are, however, scenarios in which GBN itself suffers from performance problems. In particular, when the window size and bandwidth-delay product are both large, many packets can be in the pipeline. A single packet error can thus cause GBN to retransmit a large number of packets, many of which may be unnecessary. As the probability of channel errors increases, the pipeline can .become filled with these unnecessary retransmissions. Imagine, in our message-dictation scenario, if every time a word was garbled, the surrounding 1,000 words (for example, a window size of 1,000 words) had to be repeated. The dictation would be slowed by all of the reiterated words.

As the name suggests, selective-repeat (SR) protocols avoid unnecessary retransmissions by having the sender retransmit only those packets that it suspects were received in error (that is, were lost or corrupted) at the receiver. This individual, as-needed, retransmission will require that the receiver *individually* acknowledge correctly received packets. A window size of *N* will again be used to limit the number of outstanding, unacknowledged packets in the pipeline. However, unlike GBN, the sender will have already received ACKs for some of the packets in the window. Figure 2.15 shows the SR sender's view of the sequence number space. Figure 2.16 details the various actions taken by the SR sender.

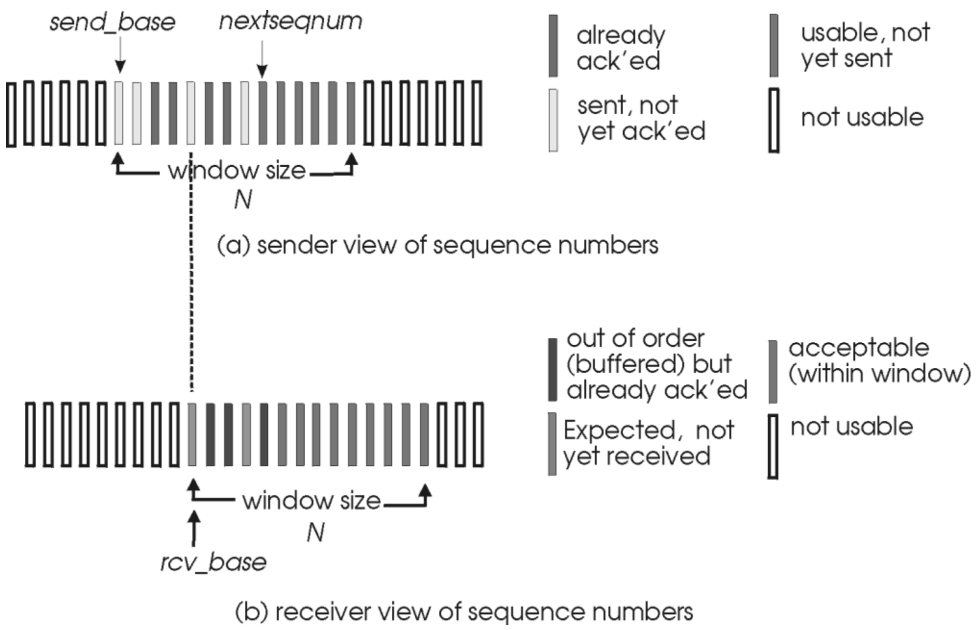

send_base
nextseqnum
window size
N
(a) sender view of sequence numbers
already ack'ed
sent, not yet ack'ed
usable, not yet sent
not usable
window size
N
rcv_base
(b) receiver view of sequence numbers
out of order (buffered) but already ack'ed
Expected, not yet received
acceptable (within window)
not usable

Figure2-15 selective repeat(SR) sender and receiver views of sequence-number

The SR receiver will acknowledge a correctly received packet whether or not it is in order. Out-of-order packets are buffered until any missing packets (that is, packets with lower sequence numbers) are received, at which point a batch of packets can be delivered in order to the upper layer. Figure 2-17 itemizes the various actions taken by the SR receiver. Figure 2-18 shows an example of SR operation in the presence of lost packets. Note that in Figure 2-18, the receiver initially buffers packets 3,4, and 5, and delivers them together with packet 2 to the upper layer when packet 2 is finally received.

It is important to note that in Step 2 in Figure 2-19, the receiver re-acknowledges (rather than ignores) already received packets with certain sequence numbers *below* the current window base. You should convince yourself that this re-acknowledgment is indeed needed. Given the sender and receiver sequence number spaces in Figure 2-15, for example, if there is no ACK for packet send_base propagating from the receiver to the sender, the sender will eventually retransmit packet send_base, even though it is clear (to us, not the sender) that the receiver has already received that packet. If the receiver were not to acknowledge this packet, the sender's window would never move forward! This example illustrates an important aspect of SR protocols (and many other protocols as well). The sender and receiver will not always have an identical view of what has been received correctly and what has not. For SR protocols, this means that the sender and receiver windows will not always coincide.

1. *Data received from above.* When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.

2. *Timeout.* Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers.

3. *ACK received.* If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to send_base, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window moves and there are un-transmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

Figure2-16 selective Repeat (SR) sender events and actions

1. *Packet with sequence number* in [rcv_base, rcv_base+N-l] is *correctly received.* In this case, the received packet falls within the receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (rcv_base in Figure 2-14), then this packet, and any previously buffered and consecutively numbered (beginning with rev_base) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer. As an example, consider Figure 2-18. When a packet with a sequence number of rev_base=2 is received, it and packets 3, 4, and 5 can be delivered to the upper layer.

2. *Packet with sequence number in* [rcv_base-N, rev_base-1] *is received.* In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.

3. *Otherwise.* Ignore the packet.

Figure 2-17 Selective Repeat (SR) receiver events and actions.

The lack of synchronization between sender and receiver windows has important consequences when we are faced with the reality of a finite range of sequence numbers. Consider what could happen, for example, with a finite range of four packet sequence numbers, 0, 1, 2, 3, arid a window size of three. Suppose packets 0 through 2 are transmitted and correctly received and acknowledged at the receiver. At this point, the receiver's window is over the fourth, fifth, and sixth packets, which have sequence numbers 3, 0, and 1, respectively. Now consider two scenarios. In the first scenario, shown in Figure 3.27a, the ACKs for the first three packets are lost and the sender retransmits these packets. The receiver thus next receives a packet with sequence number 0 a copy of the first packet sent.

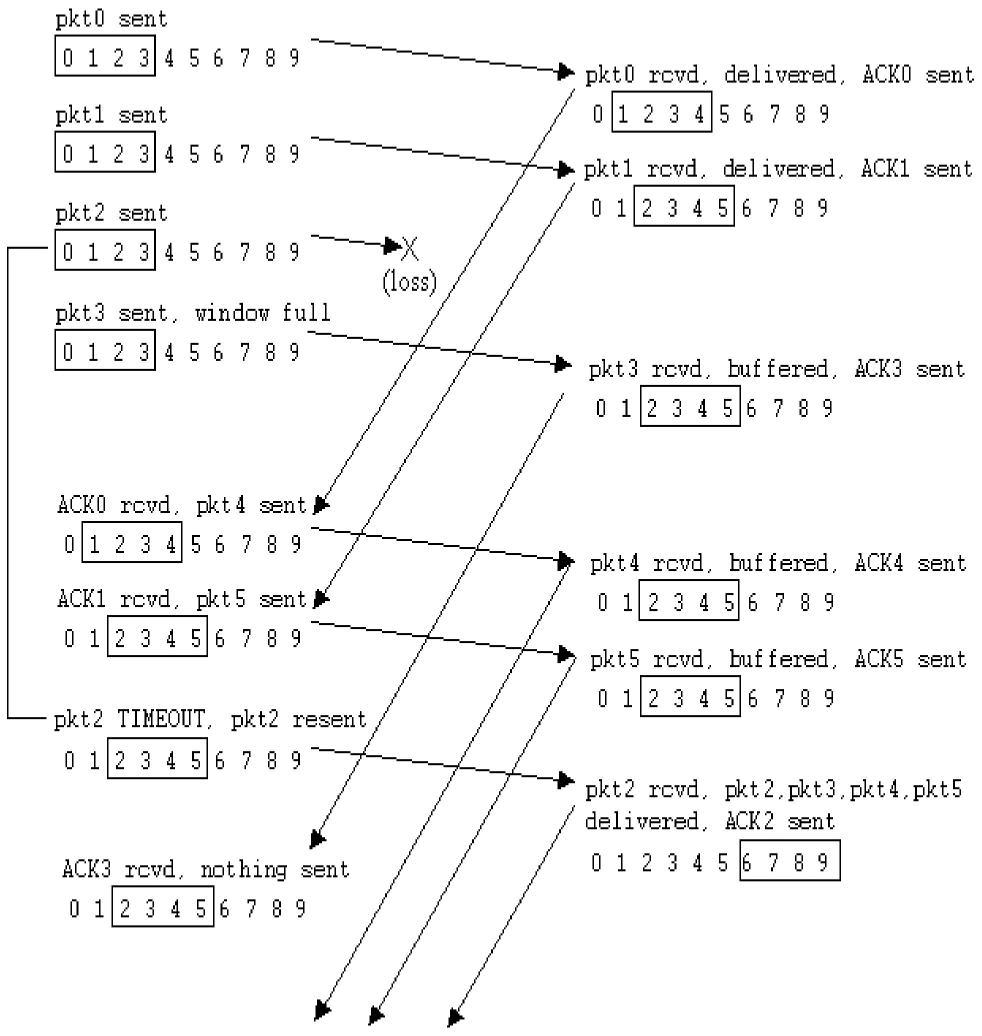


Figure2-18 SR operation

In the second scenario, shown in Figure 2.19b, the ACKs for the first three packets are all delivered correctly. The sender thus moves its window forward and sends the fourth, fifth, and sixth packets, with sequence numbers 3, 0, and 1, respectively. The packet with sequence number 3 is lost, but the packet with sequence number 0 arrivesa packet containing *new* data.

Now consider the receiver's viewpoint in Figure 2-19, which has a figurative curtain between the sender and the receiver, since the receiver cannot "see" the actions taken by the sender. The entire receiver observes is the sequence of messages it receives from the channel and sends into the channel. As far as it is concerned, the two scenarios in Figure 2-19 are *identical.* There is no way of distinguishing the retransmission of the first packet from an original transmission of the fifth packet. Clearly, a window size that is 1 less than the size of the sequence number space won't work. But how small must the window size be? A problem at the end of the chapter asks you to show that the window size must be less than or equal to half the size of the sequence number space for SR protocols.

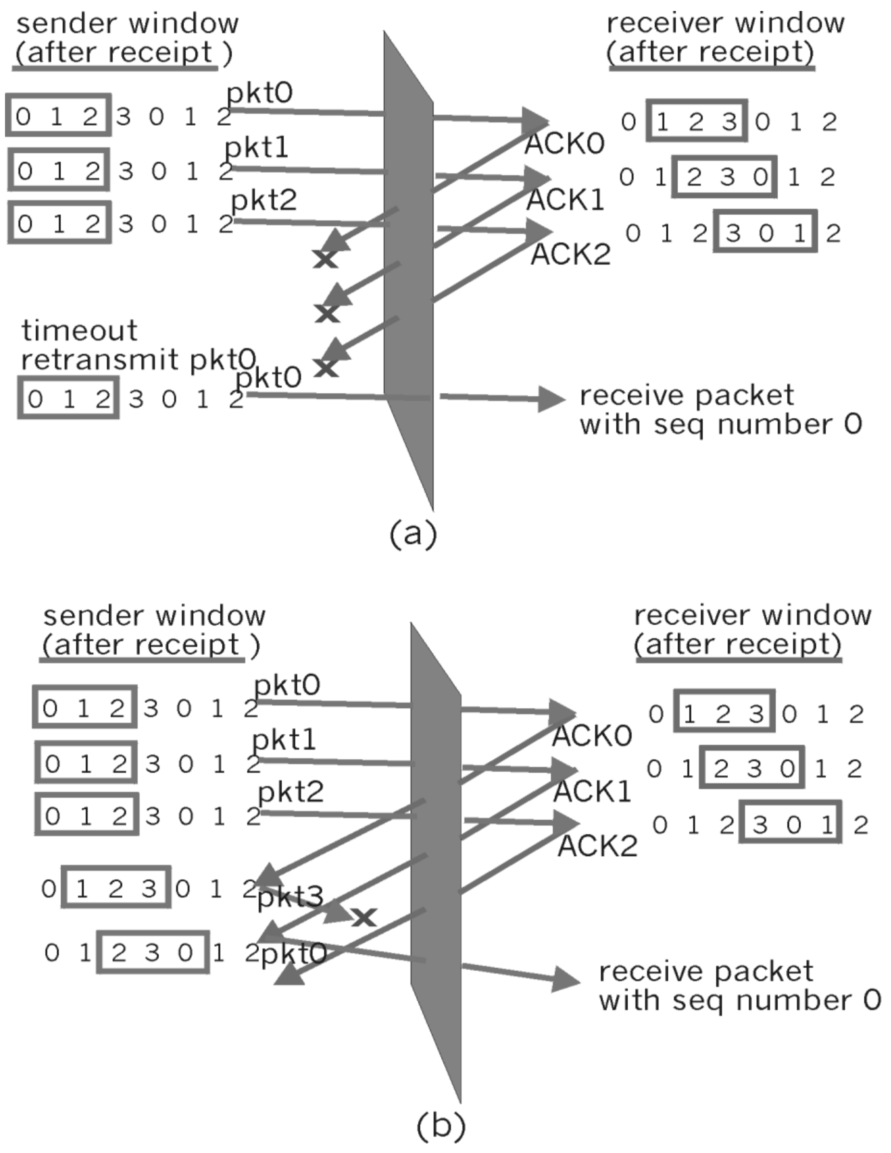


Figure2-19 SR receiver dilemma with too-large windows: A new packets or a re-transmission?

Let us conclude our discussion of reliable data transfer protocols by considering one remaining assumption in our underlying channel model. Recall that we have assumed that packets cannot be reordered within the channel between the sender and receiver. This is generally a reasonable assumption when the sender and receiver are connected by a single physical wire. However, when the "channel" connecting the two is a network, packet reordering can occur. One manifestation of packet reordering is that old copies of a packet with a sequence or acknowledgment number of can appear, even though neither the sender's nor the receiver's window contains *x.* With packet reordering, the channel can be thought of as essentially buffering packets and spontaneously emitting these packets at any point in the future. Because sequence numbers may be reused, some care must be taken to guard against such duplicate packets. The approach taken in practice is to ensure that a sequence number is not reused until the sender is "sure" that any previously sent packets with sequence number *x* are no longer in the network. This is done by assuming that a packet cannot "live" in the J network for longer than some fixed maximum amount of time. A maximum packet lifetime of approximately three minutes is assumed in the TCP extensions for high speed networks.

.

3. Connection-Oriented Transport: TCP

Structure

Objectives

- Discuss about TCP connection and segmentations
- Calculates RTT & Time out
- Discuss reliable data transfer
- Describe about flow control problems
- Understand of TCP connection managements

3-1 Introduction

Now in lesson-2 we have covered the underlying principles of reliable data transfer, let's turn to TCP the Internet's transport-layer, connection-oriented, reliable transport protocol. In this section, we'll see that in order to provide reliable data transfer, TCP relies on many of the underlying principles discussed in the previous section, including error detection, retransmissions, cumulative acknowledgments, timers, and header fields for sequence and acknowledgment numbers. TCP is defined in RFC 793, RFC 1122, RFC 1323, RFC 2018, and RFC 2581.

3-2 TCP Connection

TCP is said to be **connection-oriented** because before one application process can begin to send data to another, the two processes must first "handshake" with each other that is, they must send some preliminary segments to each other to establish the parameters of the ensuing data transfer. As part of the TCP connection establishment, both sides of the connection will initialize many TCP state variables associated with the TCP connection.

The TCP "connection" is not an end-to-end TDM or FDM circuit as in a circuit-switched network. Nor is it a virtual circuit, as the connection state resides entirely in the two end systems. Because the TCP protocol runs only in the end systems and not in the intermediate network elements (routers and bridges), the intermediate network elements do not maintain TCP connection state. In fact, the intermediate routers are completely oblivious to TCP connections; they see datagrams, not connections.

A TCP connection provides for **full-duplex** data transfer. If there is a TCP connection between process A on one host and process B on another host, then application-layer data can flow from A to B at the same time as application-layer data flows from B to A. A TCP connection is also always **point-to-point,** that is, between a single sender and a single receiver. So-called "multicasting" the transfer of data from one sender to many receivers in a single send operation is not possible with TCP. With TCP, two hosts are company and three are a crowd!

Let's now take a look at how a TCP connection is established. Suppose a process running in one host wants to initiate a connection with another process in another host. Recall that the process that is initiating the connection is called the client process, while the other process is called the server process. The client application process first informs the client transport layer that it wants to establish a connection to a process in the server. A Java client program does this by issuing the command:

```
Socket clientSocket = new Socket ("hostname", portNumber);
```

where hostname is the name of the server and portNumber identifies the process on the server. The transport layer in the client then proceeds to establish a TCP connection with the TCP in the server. We will discuss in some detail the connection establishment procedure at the end of this lesson. For now it suffices to know that the client first sends a special TCP segment; the server responds with a second special TCP segment; and finally the client responds again with a third special segment. The first two segments contain no "payload," that is, no application-layer data; the third of these segments may carry a payload. Because three segments are sent between the two hosts, this connection-establishment procedure is often referred to as a **three-way handshake.**

Once a TCP connection is established, the two application processes can send data to each other. Let us consider the sending of data from the client process to the server process. The client process passes a stream of data through the socket (the door of the process. Once the data passes through the door, the data is now in the hands of TCP running in the client. As shown in Figure 3-1, TCP directs this data to the connection's send **buffer,** which is one of the buffers that are set aside during the initial three-way handshake. From time to time, TCP will "grab" chunks of data from the send buffer. Interestingly, the TCP specification is very "laid back" about specifying when TCP should actually send buffered data, stating that TCP should "send that data in segments at its own convenience." The maximum amount of data that can be grabbed and placed in a segment is limited by **maximum segment** (MSS). The MSS depends on the TCP implementation (determined by the operating system) and can often be configured; common values are 1,500 bytes, bytes, and 512 bytes. Note that the MSS is the man mum amount of application-layer data in the segment, not the maximum size of the TCP segment including headers. (This terminology is confusing, but we have to In with it, as it is well entrenched.)

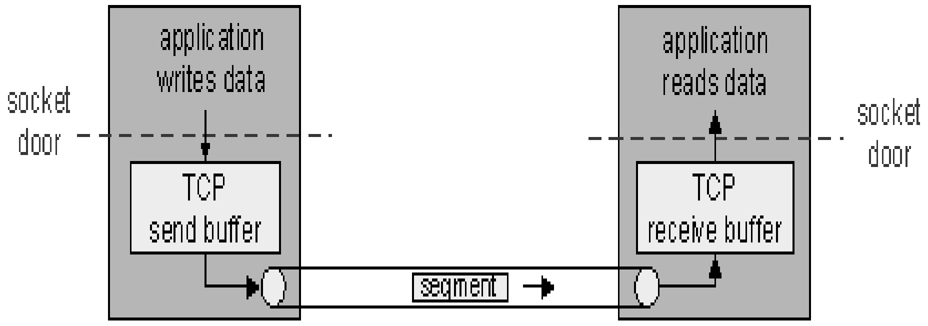


Figure3-1 TCP send and receive buffers

TCP pairs each chunk of client data with a TCP header, thereby forming TCP **segments.** The segments are passed down to the network layer, where they are separately encapsulated within network-layer IP datagrams. The IP datagrams are then sent into the network. When TCP receives a segment at the other end, the segment's data is placed in the TCP connection's receive buffer, as shown in Figure 3.1. The application reads the stream of data from this buffer. Each side of the connection has its own send buffer and its own receive buffer. (The reader is encouraged to see the online flow-control applet at http://www.awl.com/kurose-ross, which provides an animation of the send and receive buffers.)

We see from this discussion that a TCP connection consists of buffers, variables, and a socket connection to a process in one host, and another set of buffers variables, and a socket connection to a process in another host. As mentioned earlier, no buffers or variables are allocated to the connection in the network elements (routers, bridges, and repeaters) between the hosts.

3-3 TCP Segment Structure

Having taken a brief look at the TCP connection, let's examine the TCP segment structure. The TCP segment consists of header fields and a data field. The data field contains a chunk of application data. As mentioned above, the MSS limits the maximum size of a segment's data field. When TCP sends a large file, such as an image as part of a Web page, it typically breaks the file into chunks of size MSS (except for the last chunk, which will often be less than the MSS). Interactive applications, however, often transmit data chunks that are smaller than the MSS; for example, with remote login applications like Telnet, the data field in the TCP segment is often only one byte. TCP header is typically 20 bytes (12 bytes more than the UDP header), segments sent by Telnet may be only 21 bytes in length.

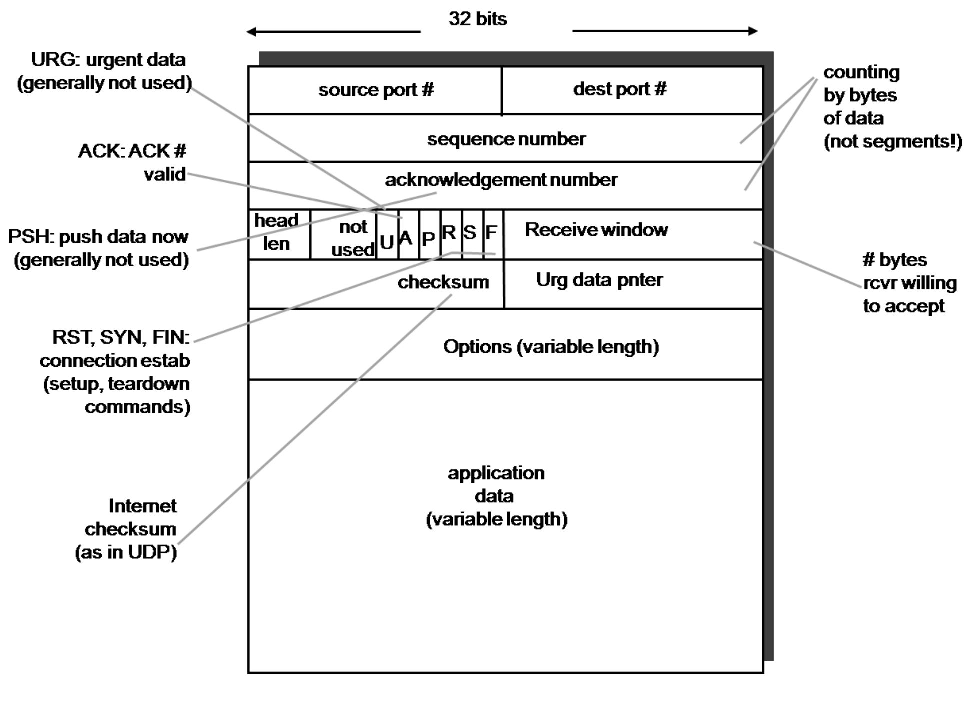
Figure 3.2 shows the structure of the TCP segment. As with UDP, the header includes **source and destination port numbers,** which are used for multiplexing/ demultiplexing data from/to upper-layer applications. Also, as with UDP, the header includes a **checksum field.** A TCP segment header also contains the following fields:

» The 32-bit **sequence number field** and the 32-bit **acknowledgment number** field are used by the TCP sender and receiver in implementing a reliable data transfer service, as discussed below.

» The 16-bit **receive window** field is used for flow control. We will see shortly that it is used to indicate the number of bytes that a receiver is willing to accept.

» The 4-bit **header length field** specifies the length of the TCP header in 32-bit words. The TCP header can be of variable length due to the TCP options field, discussed below. (Typically, the options field is empty, so that the length of the typical TCP header is 20 bytes.)

» The optional and variable-length options field is used when a sender and receiver negotiate the maximum segment size (MSS) or as a window scaling factor for use in high-speed networks. A time-stamping option is also defined. See RFC is 854 and RFC 1323 for additional details.


32 bits
URG: urgent data (generally not used)
ACK: ACK # valid
PSH: push data now (generally not used)
RST, SYN, FIN: connection estab (setup, teardown commands)
Internet checksum (as in UDP)
source port #
dest port #
sequence number
acknowledgement number
head len
not used
U
A
P
R
S
F
Receive window
checksum
Urg data pnter
Options (variable length)
application data (variable length)
counting by bytes of data (not segments!)
# bytes rcvr willing to accept

» The **flag field** contains 6 bits. The **ACK bit** is used to indicate that the value carried in the acknowledgment field is valid. The **RST,** SYN, and **FIN** bits are used for connection setup and teardown, as we will discuss at the end of this section. When the **PSH** bit is set, this is an indication that the receiver should pass tl data to the upper layer immediately. Finally, the **URG** bit is used to indicate that there is data in this segment that the sending-side upper-layer entity has marked) as "urgent." The location of the last byte of this urgent data is indicated by the 16-bit **urgent data pointer field.** TCP must inform the receiving-side upper layer entity when urgent data exists and pass it a pointer to the end of the urgent data. (In practice, the PSH, URG, and the urgent data pointer are not used. However, we mention these fields for completeness.)

3-3-1 Sequence Numbers and Acknowledgment Numbers

Two of the most important fields in the TCP segment header are the sequence number field and the acknowledgment number field. These fields are a critical part of TCP'S reliable data transfer service. But before discussing how these fields are used to provide reliable data transfer, let us first explain what exactly TCP puts in these fields.

TCP views data as an unstructured, but ordered, stream of bytes. TCP's use of sequence numbers reflects this view in that sequence numbers are over the stream of transmitted bytes and not over the series of transmitted segments. The **sequence number for a segment** is therefore the byte-stream number of the first byte in the segment. Let's look at an example. Suppose that a process in host A wants to send a stream of data to a process in host B over a TCP connection. The TCP in host A will implicitly number each byte in the data stream. Suppose that the data stream consists of a file consisting of 500,000 bytes, that the MSS is 1,000 bytes, and that the first byte of the data stream is numbered zero. As shown in Figure 3.3, TCP constructs 500 segments out of the data stream. The first segment gets assigned sequence number 0, the second segment gets assigned sequence number 1000, and the third segment gets assigned sequence number 2000, and so on. Each sequence number is inserted in the sequence number field in the header of the appropriate TCP segment.

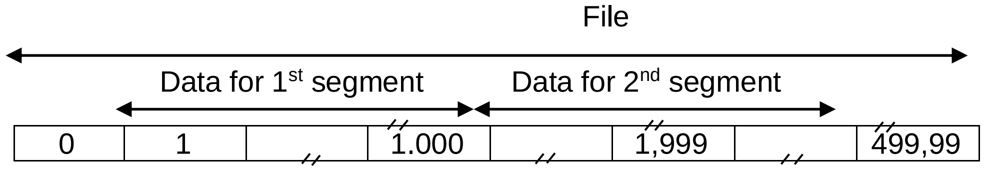


Figure 3-3 dividing file data into TCP segments

Now let us consider acknowledgment numbers. These are a little trickier than sequence numbers. Recall that TCP is full-duplex, so that host A may be receiving data from host B while it sends data to host B (as part of the same TCP connection). Each of the segments that arrive from host B has a sequence number for the data flowing from B to A. *The acknowledgment number that host-A puts in its segment is the sequence number of the next byte host A is expecting from host B.* It is good to look at a few examples to understand what is going on here. Suppose that host A has received all bytes numbered 0 through 535 from B and suppose that it is about to send a segment to host B. Host A is waiting for byte 536 and all the subsequent bytes in host B's data stream. So host A puts 536 in the acknowledgment number field of the segment it sends to B.

As another example, suppose that host A has received one segment from host B containing bytes 0 through 535 and another segment containing bytes 900 through 1,000. For some reason host A has not yet received bytes 536 through 899. In this example, host A is still waiting for byte 536 (and beyond) in order to recreate B's data stream. Thus, A's next segment to B will contain 536 in the acknowledgment number field. Because TCP only acknowledges bytes up to the first missing byte in the stream, TCP is said to provide **cumulative acknowledgments.**

This last example also brings up an important but subtle issue. Host A received the third segment (bytes 900 through 1,000) before receiving the second segment (bytes 536 through 899). Thus, the third segment arrived out of order. The subtle issue is: what does a host do when it receives out-of-order segments in a TCP connection? Interestingly, the TCP RFCs do not impose any rules here and leave the decision up to the people programming a TCP implementation. There are basically two choices: either (1) the receiver immediately discards out-of-order segments, or (2) the receiver keeps the out-of-order bytes and waits for the missing bytes to fill in the gaps. Clearly, the latter choice is more efficient in terms of network bandwidth, whereas the former choice simplifies the TCP code.

In Figure 3.3 we assumed that the initial sequence number was zero. In truth, both sides of a TCP connection randomly choose an initial sequence number. This is done to minimize the possibility that a segment that is still present in the network from an earlier, already-terminated connection between two hosts is mistaken for a valid segment in a later connection between these same two hosts (which also happen to be using the same port numbers as the old connection).

3-3-2 Telnet: A Case Study for Sequence and Acknowledgment Numbers

Telnet, defined in RFC 854, is a popular application-layer protocol used for remote login. It runs over TCP and is designed to work between any pair of hosts. Unlike the bulk data transfer applications discussed in Chapter 2, Telnet is an interactive application. We discuss a Telnet example here, as it nicely illustrates TCP sequence and acknowledgment numbers.

Suppose host A initiates a Telnet session with host B. Because host A initiates the session, it is labeled the client, and host B is labeled the server. Each character typed by the user (at the client) will be sent to the remote host; the remote host will send back a copy of each character, which will be displayed on the Telnet user's screen. This "echo back" is used to ensure that characters seen by the Telnet user have already been received and processed at the-remote site. Each character thus traverses the network twice between the time the user hits the key and the time character is displayed on the user's monitor.

Now suppose the user types a single letter, 'C,' and then grabs a coffee. Let's examine the TCP segments that are sent between the client and server. As shown in Figure 3.4, we suppose the starting sequence numbers are 42 and 79 for the client and server, respectively. Recall that the sequence number of a segment is the sequence number of the first byte in the data field. Thus, the first segment sent from the client will have sequence number 42; the first segment sent from the server will have sequence number 79. Recall that the acknowledgment number is the sequence number of the next byte of data that the host is waiting for. After the TCP connection is established but before any data is sent, the client is waiting for byte 79 and the server is waiting for byte 42.

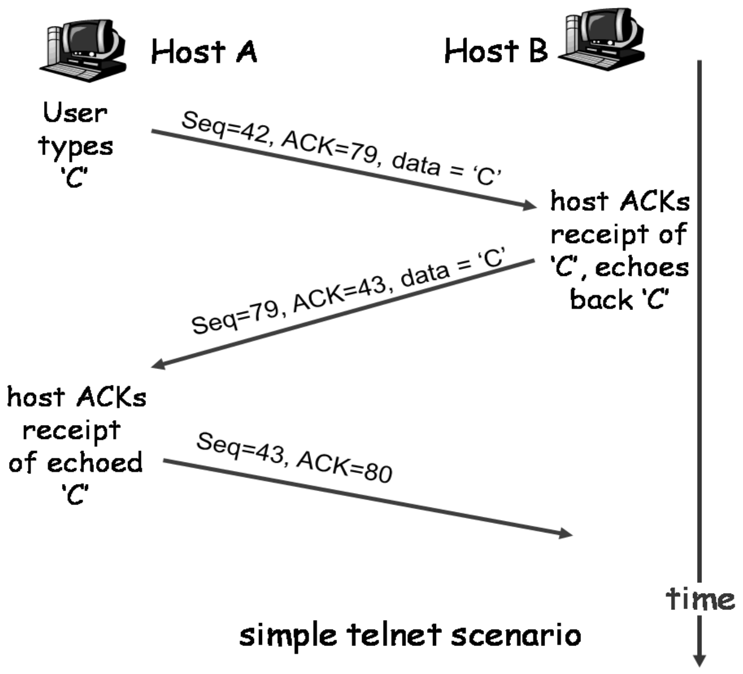

Host A
Host B
User types 'C'
Seq=42, ACK=79, data = 'C'
host ACKs receipt of 'C', echoes back 'C'
Seq=79, ACK=43, data = 'C'
host ACKs receipt of echoed 'C'
Seq=43, ACK=80
time
simple telnet scenario

Figure 3-4 Sequence and acknowledgement numbers for a simple Telnet application over TCP

As shown in Figure 3.4, three segments are sent. The first segment is sent ft the client to the server, containing the one-byte ASCII representation of the letter 'C' in its data field. This first segment also has 42 in its sequence number field, as we just described. Also, because the client has not yet received any data from the server, this first segment will have 79 in its acknowledgment number field.

The second segment is sent from the server to the client. It serves a dual purpose. First it provides an acknowledgment of the data the server has received. By putting 43 in the acknowledgment field, the server is telling the client that it has successfully received everything up through byte 42 and is now waiting for bytes 43 onward. The second purpose of this segment is to echo back the letter 'C.' Thus, the second segment has the ASCII representation of 'C' in its data field. This second segment has the sequence number 79, the initial sequence number of the server-to-client data flow of this TCP connection, as this is the very first byte of data that the server is sending. Note that the acknowledgment for client-to-server data is carried in a segment carrying server-to-client data; this acknowledgment is said to be **piggybacked** on the server-to-client data segment.

The third segment is sent from the client to the server. Its sole purpose is to acknowledge the data it has received from the server. (Recall that the second segment contained data the letter 'C' from the server to the client.) This segment has an empty data field (that is, the acknowledgment is not being piggybacked with any client-to-server data). The segment has 80 in the acknowledgment number field because the client has received the stream of bytes up through byte sequence number 79 and it is now waiting for bytes 80 onward. You might think it odd that this segment also has a sequence number since the segment contains no data. But because TCP has a sequence number field, the segment needs to have some sequence number.

3.4 Round-Trip Time Estimation and Timeout

We will soon see that TCP, like our rdt protocol in lesson-2, uses a timeout/retransmit mechanism to recover from lost segments. Although conceptually simple, many subtle issues arise when implementing a timeout/retransmit mechanism in an actual protocol such as TCP. Perhaps the most obvious question is the length of the timeout intervals. Clearly, the timeout should be larger than the connection's round-trip time (RTT), that is, the time from when a segment is sent until it is acknowledged. Otherwise, unnecessary retransmissions would be sent. But how much larger? How should the RTT be estimated in the first place? Should a timer be associated with each and every unacknowledged segment? So many questions! Our discussion in this section is based on the TCP work in and the current IETF recommendations for managing TCP timers [RFC 2988].

3-4-1 Estimating the Round-Trip Time

Let's begin our study of TCP timer management by considering how TCP estimates the round-trip time between sender and receiver. This is accomplished as follows. The sample RTT, denoted SampleRTT, for a segment is the amount of time from when ' the segment is sent (that is, passed to IP) until an acknowledgment for the segment is received. Instead of measuring a SampleRTT for every transmitted segment, most TCP implementations take only one SampleRTT measurement at a time. That is, at any point in time, the SampleRTT is being estimated for only one of the transmitted but currently unacknowledged segments, leading to a new value of SampleRTT approximately once every round-trip time. Also, TCP never computes a SampleRTT for a segment that has been retransmitted; it only measures SampleRTT for segments that have been transmitted once.

Obviously, the SampleRTT values will fluctuate from segment to segment due to congestion in the routers and to the varying load on the end systems. Because of this fluctuation, any given SampleRTT value may be atypical. In order to estimate a typical RTT, it is therefore natural to take some sort of average of the SampleRTT values. TCP maintains an average, called EstimatedRTT, of the SampleRTT values. Upon obtaining a new SampleRTT, TCP updates EstimatedRTT according to the following formula:

$$\text{EstimatedRTT} = (I - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$$

The formula above is written in the form of a programming-language statement the new value of EstimatedRTT is a weighted combination of the previous value of EstimatedRTT and the new value for SampleRTT. The recommended value of α is $\alpha = 0.125$ (that is, 1/8) [RFC 2988], in which case the formula above becomes:

$$\text{EstimatedRTT} = 0.875 \cdot \text{EstimatedRTT} + 0.125 \cdot \text{SampleRTT}$$

Note that EstimatedRTT is a weighted average of the SampleRTT values. This weighted average puts more weight on recent samples than on old samples. This is natural, as the more recent samples better reflect the current congestion in the network. In statistics, such an average is called an **exponential weighted moving average** (EWMA). The word "exponential" appears in EWMA because the weight of a given SampleRTT decays exponentially fast as the updates proceed. In the homework prob-lems you will be asked to derive the exponential term in EstimatedRTT.

Figure 3.5 shows the SampleRTT values and EstimatedRTT for a value of $\alpha = 1/8$ for a TCP connection between gaia.cs.umass.edu (in Amherst, Massachusetts) to fantasia.eurecom.fr (in the south of France). Clearly, the variations in the SampleRTT are smoothed out in the computation of the EstimatedRTT.

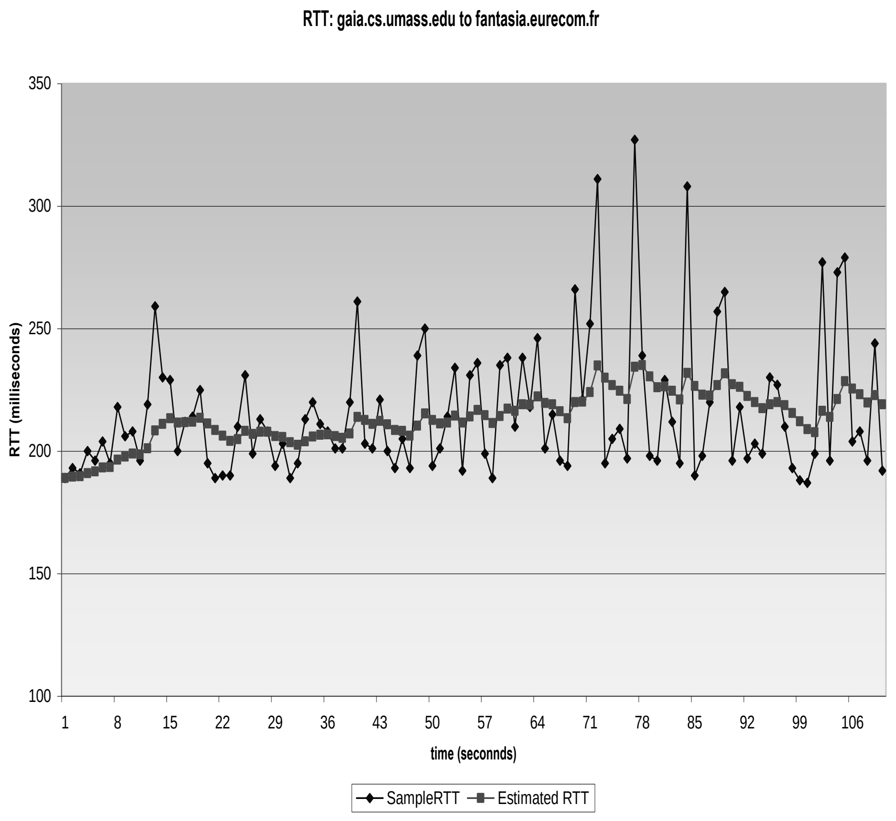


Figure 3-5 RTT samples and RTT estimates

In addition to having an estimate of the RTT, it is also valuable to have a measure of the variability of the RTT. [RFC 2988] defines the round-trip-time variation, DevRTT, as an estimate of how much SampleRTT typically deviates from EstimatedRTT:

$$\text{DevRTT} = (1 - \beta) \cdot \text{DevRTT} + \beta \cdot |\ \text{SampleRTT} - \text{EstimatedRTT}\ |$$

Note that DevRTT is an EWMA of the difference between SampleRTT and EstimatedRTT. If the SampleRTT values have little fluctuation, then DevRTT will be small; on the other hand, if there is a lot of fluctuation, DevRTT will be large. The recommended value β is 0.25.

3-4-2 Setting and Managing the Retransmission Timeout Interval

Given values of EstimatedRTT and DevRTT, what value should be used for TCP's timeout interval? Clearly, the interval should be greater than or equal to EstimatedRTT. Otherwise, unnecessary retransmissions would be sent. But the timeout interval should not be too much larger than EstimatedRTT; otherwise, when a segment is lost, TCP would not quickly retransmit the segment, thereby introducing significant data transfer delays into the application. It is therefore desirable to set the timeout equal to the EstimatedRTT plus some margin. The margin should be large when there is a lot of fluctuation in the SampleRTT values; it should be small when there is little fluctuation. The value of DevRTT should thus come into play here. All of these considerations are taken into account in TCP's method for determining the retransmission timeout interval:

$$\text{TimeoutIntetval} = \text{EstimtedRTT} + 4 \cdot \text{DevRTT}$$

3-5 Reliable Data Transfer

Internet's network-layer service (IP service) is unreliable. IP does not guarantee datagram delivery, does not guarantee in-order delivery of datagrams, and does not guarantee the integrity of the data in the datagrams. With IP service, datagrams can overflow router buffers and never reach their destination, datagrams can arrive out of order, and bits in the datagram can get corrupted (flipped from 0 to 1 and vice versa). Because transport-layer segments are carried across the network by IP datagrams, transport-layer segments can suffer from these problems as well.

TCP creates a **reliable data transfer service** on top of IP's unreliable best-effort service. TCP's reliable data transfer service ensures that the data stream that a process reads out of its TCP receive buffer is uncorrupted, without gaps, without duplication, and in sequence; that is, the byte stream is exactly the same byte stream that was sent by the end system on the other side of the connection. In this subsection, we provide an overview of how TCP provides a reliable data transfer. We'll see that the reliable data transfer service of TCP uses many of the principles that we studied in lesson-2.

In our earlier development of reliable data transfer techniques in lesson-2, it was conceptually easiest to assume that an individual timer is associated with each transmitted but not yet acknowledged segment. While this is great in theory, timer management can require considerable overhead. Thus, the recommended TCP timer management procedures use only a *single* retransmission timer, even if there are multiple transmitted but not yet acknowledged segments. The TCP protocol provided in this section follows this single-timer recommendation.

We will discuss how TCP provides reliable data transfer in two incremental steps. We first present a highly simplified description of a TCP sender that uses only timeouts to recover from lost segments; we then present a more complete description that uses duplicate acknowledgments in addition to timeouts. In the ensuing discussion, we suppose that data is being sent in only one direction, from host A to host B, and that host A is sending a large file.

Figure 3.6 presents our highly simplified description of a TCP sender. We see that there are three major events related to data transmission and retransmission in the TCP sender: data received from application above; timer timeout; and ACK received. Upon the occurrence of the first major event, TCP receives data from the application, encapsulates the data in a segment, and passes the segment to IP. Note that each segment includes a sequence number that is the byte-stream number of the first data byte in the segment, as described in Section 3-3. Also note, if the timer is already not running for some other segment, TCP starts the timer when the segment is passed to IP. (It is helpful to think of the timer as being associated with the' oldest unacknowledged segment.) The expiration interval for this timer is the TimeoutInterval, which is calculated from EstimatedRTT and DevRTT,| as described in Section 3.4. ;

/* Assume sender is not constrained by TCP flow or congestion control, that data from above is less than MSS in size, and that data transfer is in one direction only. */

```
NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (forever) {
   switch(event)

      event: data received from application above
         create TCP segment with sequence number NextSeqNum
         if (timer currently not running)
            start timer
         pass segment to IP
         NextSeqNum=NextSeqNum+length(data)
         break;

      event: timer timeout
         retransmit not-yet-acknowledged segment with
            smallest sequence number
         start timer
         break;

      event: ACK received, with ACK field value of y
         if (y > SendBase) {
            SendBase=y
            if (there are currently any not-yet-acknowledged segments)
               start timer
            }
            break;

   } /* end of loop forever */
```

Figure 3-6 Simplified TCP sender

The second major event is the timeout event. TCP responds to the timeout event by retransmitting the segment that caused the timeout. TCP then restarts the timer.

The third major event that must be handled by the TCP sender is the arrival of M acknowledgment segment (ACK) from the receiver (more specifically, a segment containing a valid ACK field value). On the occurrence of this event, TCP compares the ACK value y with its variable SendBase. The TCP state variable SendBase is the sequence number of the oldest unacknowledged byte. (Thus SendBase-1 is the sequence number of the last byte that is known to have been received con and in order at the receiver.) As indicated earlier, TCP uses cumulative acknowla_ ments, so that y acknowledges the receipt of all bytes before byte number y. if y > SendBase, then the ACK is acknowledging one or more previously mil knowledged segments. Thus the sender updates its SendBase variable; it also restarts the timer if there currently are any not yet acknowledged segments.

3-5-1 A Few Interesting Scenarios

We have just described a highly simplified version of how TCP provides reliable data transfer. But this highly simplified version although simplified still has many subtleties. To get a good feeling of how this protocol works, let's now walk through a few simple scenarios. Figure 3.7 depicts the first scenario, in which host A sends one segment to host B. Suppose that this segment has sequence number 92 and contains eight bytes of data. After sending this segment, host A waits for a segment from B with acknowledgment number 100. Although the segment from A is received at B, the acknowledgment from B to A gets lost. In this case, the time out event occurs, and host A retransmits the same segment. Of course, when host B receives the retransmission, it will observe from the sequence number that the segment contains data that has already been received. Thus, TCP in host B will discard the bytes in the retransmitted segment.

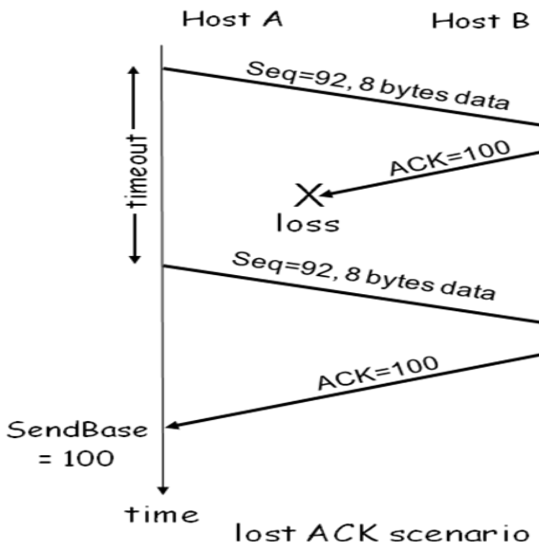


Figure 3-7 Retransmissions due to a lost acknowledgment

In a second scenario, shown in Figure 3.8, host A sends two segments back to back. The first segment has sequence number 92 and eight bytes of data, and the second segment has sequence number 100 and 20 bytes of data. Suppose that both segments arrive intact at B, and B sends two separate acknowledgments for each of these segments. The first of these acknowledgments has acknowledgment number 100; the second has acknowledgment number 120. Suppose now that neither of the acknowledgments arrives at host A before the timeout. When the timeout event occurs, host A resends the first segment with sequence number 92 and restarts the timer. As long as the ACK for the second segment arrives before the new timeout, the second-segment will not be retransmitted.

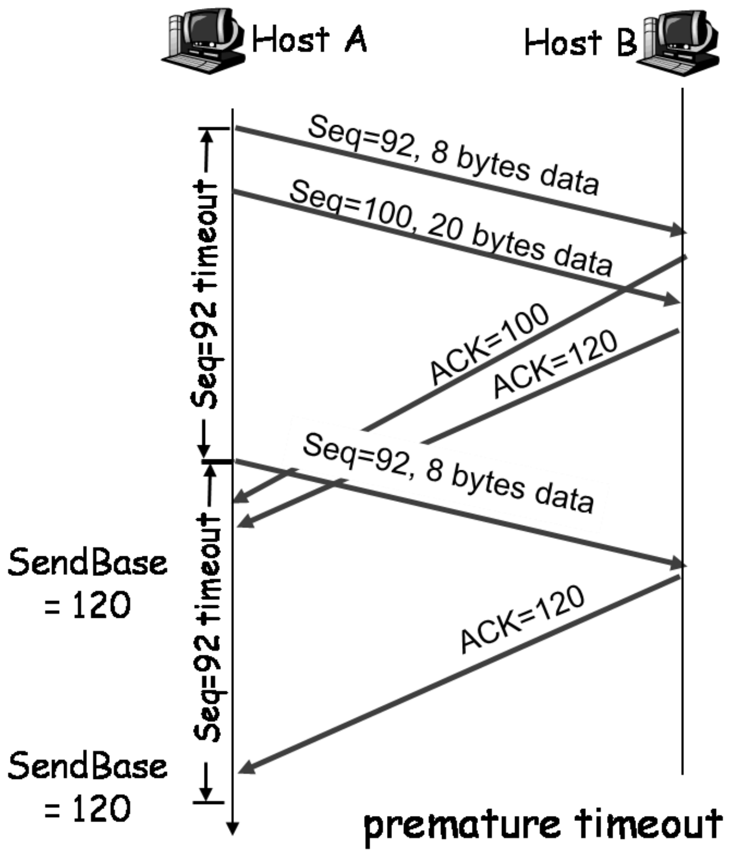


Figure 3-8 Segment 100 not retransmitted

In a third and final scenario, suppose host A sends the two segments, exactly as in the second example. The acknowledgment of the first segment is lost in the network, but just before the timeout event, host A receives an acknowledgment with acknowledgment number 120. Host A therefore knows that host B has received *everything* up through byte 119; so host A does not resend either of the two segments. This scenario is illustrated in Figure 3.9.

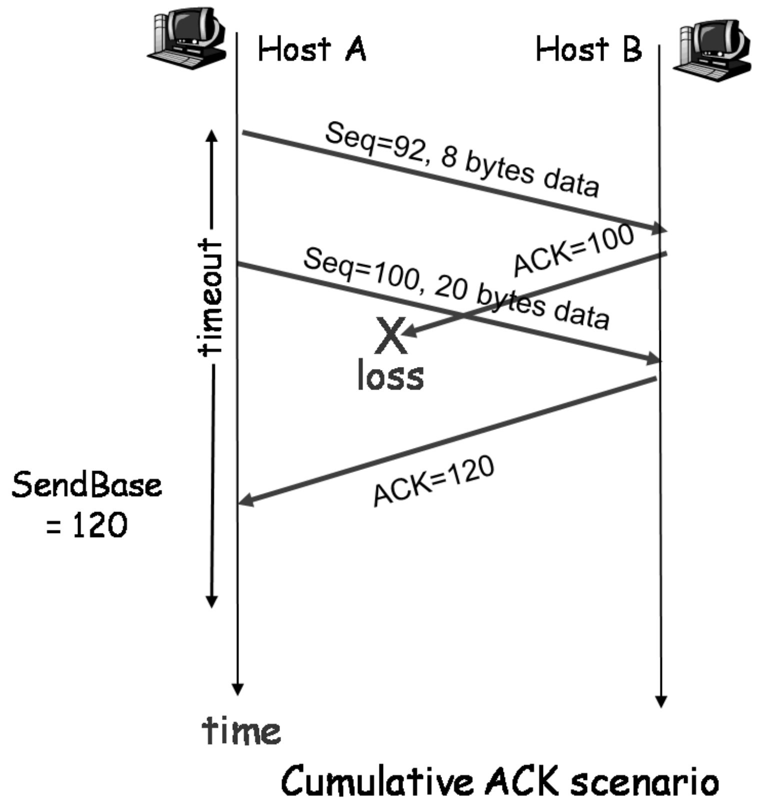


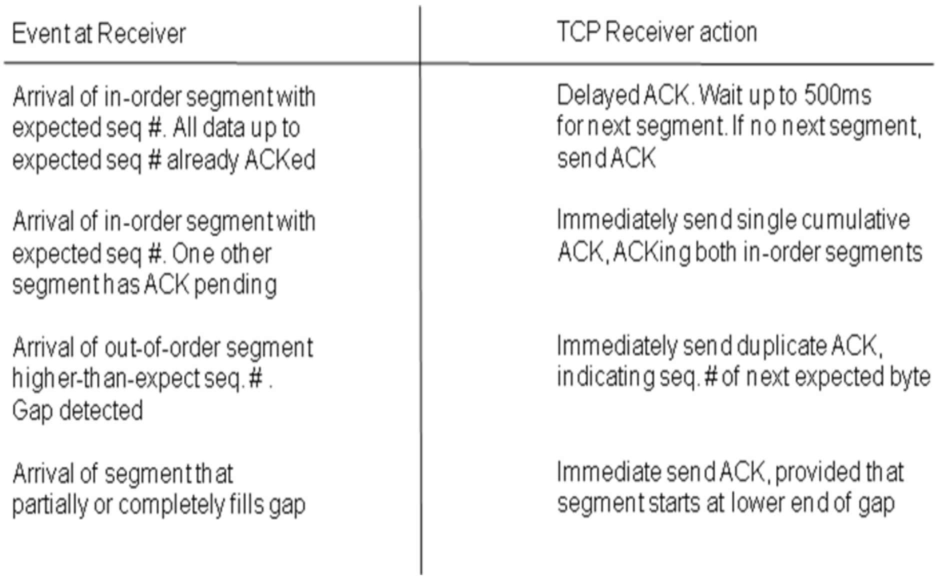
Figure 3-9 A cumulative acknowledgment avoids retransmission of the first segment

3-5-2 Doubling the Timeout Interval

We now discuss a few modifications that most TCP implementations employ. The first concerns the length of the timeout interval after timer expiration. In this modification, whenever the timeout event occurs, TCP retransmits the not yet acknowledged segment with the smallest sequence number, as described above. But each time TCP retransmits, it sets the next timeout interval to twice the previous value rather than deriving it from the last EstimatedRTT and DevRTT (as described in Section 3.4). For example, suppose TimeoutInterval associated with the oldest not yet acknowledged segment is .75 sec when the timer first expires. TCP will then retransmit this segment and set the new expiration time to 1.5 sec. If the timer expires again 1.5 seconds later, TCP will again retransmit this segment, now setting the expiration time to 3.0 seconds. Thus the intervals grow exponentially after each retransmission. However, whenever the timer is started after either of the two other events (that is, data received from application above, and ACK received), the TimeoutInterval is derived from the most recent values of EstimatedRTT, and DevRTT.

3-5-3 Fast Retransmit

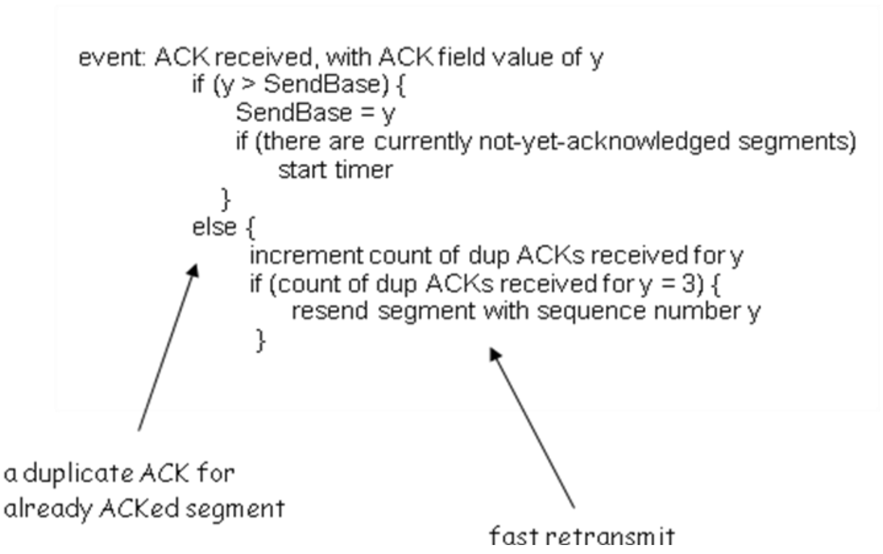
One of the problems with timeout-triggered retransmissions is that the timeout period is often relatively long. When a segment is lost, this long timeout period forces the sender to wait a long time before resending the lost packet, thereby increasing the end-to-end delay. Fortunately, the sender can often detect packet loss well before the timeout event occurs by noting so-called duplicate ACKs. A. duplicate **ACK** is an ACK that re-acknowledges a segment for which the sender has already received an earlier acknowledgment. To understand the sender's response to a du-plicate ACK, we must look at why the receiver sends a duplicate ACK in the first place. Table 3.1 summarizes the TCP receiver's ACK generation policy. When a TCP receiver receives a segment with a sequence number that is larger than the next, expected, in-order sequence number, it detects a gap in the data stream that is, a missing segment. This gap could be the result of lost or reordered segments within the network. Since TCP does not use negative acknowledgments, the receiver cannot send an explicit negative acknowledgment back to the sender. Instead, it simply re-acknowledges (that is, generates a duplicate ACK for) the last in-order byte of data it has received. (Note that Table 3.1 allows for the case that the receiver does not discard out-of-order segments.)



Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send duplicate ACK, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment starts at lower end of gap

Table 3-1 TCP ACK Generation Recommendation

Because a sender often sends a large number of segments back to back, if one segment is lost, there will likely be many back-to-back duplicate ACKs. If the TCP sender receives three duplicate ACKs for the same data, it takes this as an indication that the segment following the segment that has been ACKed three times has been lost. (In the homework problems, we consider the question of why the sender waits for three duplicate ACKs, rather than just a single duplicate ACK.) In case the three duplicate ACKs are received, TCP performs a fast **retransmit**, retrans-mitting the missing segment *before* that segment's timer expires. For TCP with fast retransmit, the following code snippet replaces the ACK received event in Figure 3.6:



```
event: ACK received, with ACK field value of y
          if (y > SendBase) {
              SendBase = y
              if (there are currently not-yet-acknowledged segments)
                  start timer
          }
          else {
               increment count of dup ACKs received for y
               if (count of dup ACKs received for y = 3) {
                   resend segment with sequence number y
               }
```

We noted earlier that many subtle issues arise when implementing a timeout/retransmit mechanism in an actual protocol such as TCP. The procedures above, which have evolved as a result of more than 15 years of experience with TCP timers, should convince the reader that this is indeed the case!

3-5-4 Go-Back-N or Selective Repeat?

Let us close our study on TCP's error-recovery mechanism by considering the following question: is TCP a GBN or an SR protocol? Recall that TCP acknowledgments are cumulative and correctly received but out-of-order segments are not individually ACKed by the receiver. Consequently, as shown in Figure 3.6 (see also Figure 2-11 in lesson-2), the TCP sender need only maintain the smallest sequence number of a transmitted but unacknowledged byte (SendBase) and the sequence number of the next byte to be sent (NextSeqNum). In this sense, TCP looks a lot like a GBN-style protocol. But there are some striking differences between TCP and Go-Back-N. Many TCP implementations will buffer correctly received but out-of-order segments. Consider also what happens when the sender sends a sequence of segments 7, 2,..., M and all of the segments arrive in order without error at the receiver. Further suppose that the acknowledgment for packet $n < N$ gets lost, but the remaining $N - 1$ acknowledgments arrive at the sender before their respective timeouts. In this example, Go-Back-N would retransmit not only packet *n,* but also all of the subsequent packets .*n* + 1, *n + 2,..., N.* TCP, on the other hand, would retransmit at most, one segment, ' namely, segment *n.* Moreover, TCP would not even retransmit segment *n* if the acknowledgment for segment *n* + 1 arrived before the timeout for segment *n.*

A proposed modification to TCP, the so-called **selective acknowledgment**, allows a TCP receiver to acknowledge out-of-order segments selectively rather than cumulatively acknowledging the last correctly received, in-order segment. When combined with selective retransmission skipping the retransmission of segments that have already been selectively acknowledged by the receiver TCP looks a lot like our generic SR protocol. Thus, TCP's error-recovery mechanism is probably best categorized as a hybrid of Go-Back-N and selective-repeat J protocols.

3-6 Flow Control

Recall that the hosts on each side of a TCP connection set aside a receive buffer for I the connection. When the TCP connection receives bytes that are correct and in sequence, it places the data in the receive buffer. The associated application process will read data from this buffer, but not necessarily at the instant the data arrives. Indeed, the receiving application may be busy with some other task and may not even attempt to read the data until long after it has arrived. If the application is relatively slow at reading the data, the sender can very easily overflow the connections receive buffer by sending too much data too quickly.

TCP provides a **flow-control service** to its applications to eliminate the possibility of the sender overflowing the receiver's buffer. Flow control is thus a speed matching service matching the rate at which the sender is sending to the rate at I which the receiving application is reading. As noted earlier, a TCP sender can also be throttled due to congestion within the IP network; this form of sender control is referred to as **congestion control**. Even though the actions taken by flow and congestion control are similar (the throttling of the sender), they are obviously taken for very different reasons. Unfortunately, many authors use the term interchangeably, and the savvy reader would be careful to distinguish between the two cases. Let's now discuss how TCP provides its flow-control service. In order to see the forest for the trees, we suppose throughout this section that the TCP implementation is such that the TCP receiver discards out-of-order segments.

TCP provides flow control by having the sender maintain a variable called the **receive window.** Informally, the-receive window is used to give the sender an idea of how much free buffer space is available at the receiver. Because TCP is full-duplex, the sender at each side of the connection maintains a distinct receive window. Let's investigate the receive window in the context of a file transfer. Suppose that host A is sending a large file to host B over a TCP connection. Host B allocates a receive buffer to this connection; denote its size by RcvBuffer. From time to time, the application process in host B reads from the buffer. Define the following variables:

• LastByteRed: the number of the last byte in the data stream read from the buffer by the application process in B

• LastByteRcvd: the number of the last byte in the data stream that has arrived from the network and has been placed in the receive buffer at B

Because TCP is not permitted to overflow the allocated buffer, we must have:

$$\text{LastByteRcvd} - \text{LastByteRead} \leq \text{RcvBuffer}$$

The receive window, denoted RcvWindow, is set to the amount of spare room in the buffer:

$$\text{RcvWindow} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]$$

Because the spare room changes with time, RcvWindow is dynamic. The variable RcvWindow is illustrated in Figure 3.10.

How does the connection use the variable RcvWindow to provide the flow-control service? Host B tells host A how much spare room it has in the connection buffer by placing its current value of RcvWindow in the receive window field of every segment it sends to A. Initially, host B sets RcvWindow = RcvBuffer. Note that to pull this off, host B must keep track of several connection-specific variables.

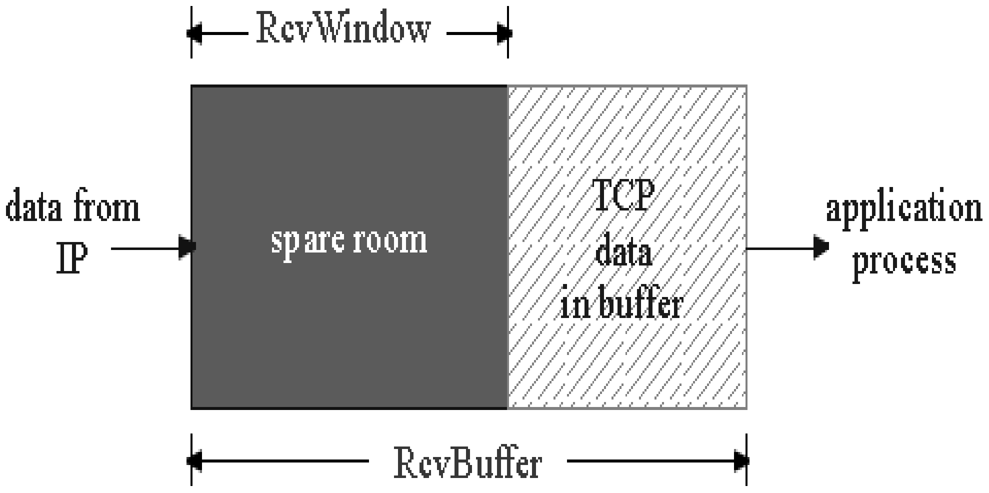


Figure 3-10 the receive window (RcvWindow) and the receive buffer (RcvBuffer)

Host A in turn keeps track of two variables, LastByteSent and LastByteAcked, which have obvious meanings. Note that the difference between these two variables, LastByteSent - LastByteAcked, is the amount of unacknowledged data that A has sent into the connection. By keeping the amount of unacknowledged data less than the value of RcvWindow, host A is assured that it is not overflowing the receive buffer at host B. Thus, host A makes sure throughout the connection's life that

$$\text{LastByteSent} - \text{LastByteAcked} < \text{RcvWindow}$$

There is one minor technical problem with this scheme. To see this, suppose host B.'s receive buffer becomes full so that RcvWindow = 0. After advertising RcvWindow = 0 to host A, also suppose that B has *nothing* to send to A. Now consider what happens. As the application process at B empties the buffer, TCP does not send new segments with new RcvWindow values to host A; indeed, TCP sends a segment to host A only if it has data to send or if it has an acknowledgment to send. Therefore, host A is never informed that some space has opened up in host B's receive buffer host A is blocked and can transmit no more data! To solve this problem, the TCP specification requires host A to continue to send segments with one data byte when B's receive window is zero. These segments will be acknowledged by the receiver. Eventually the buffer will begin to empty and the acknowledgments will contain a nonzero RcvWindow value.

Having described TCP's flow-control service, we briefly mention here that UDP does not provide flow control. To understand the issue here, consider sending a series of UDP segments from a process on host A to a process on host B. For a typical UDP implementation, UDP will append the segments in a finite-sized buffer that "precedes" the corresponding socket (that is, the door to the process). The process reads one entire segment at a time from the buffer. If the process does not read the segments fast enough from the buffer, the buffer will overflow and segments will get dropped.

3-7 TCP Connection Management

In this subsection we take a closer look at how a TCP connection is established and torn down. Although this topic may not seem particularly thrilling, it is important because TCP connection establishment can significantly add to perceived delays (for example, when surfing the Web). Let's now take a look at how a TCP connection is established. Suppose a process running in one host (client) wants to initiate a connection with another process in another host (server). The client application process first informs the client TCP that it wants to establish a connection to a process in the server. The TCP in the client then proceeds to establish a TCP connection with the TCP in the server in the following manner:

- **Step 1.** The client-side TCP first sends a special TCP segment to the server-side TCP. This special segment contains no application-layer data. But one of the flag bits in the segment's header (see Figure 3.2), the SYN bit, is set to 1. For this reason, this special segment is referred to as a SYN

segment. In addition, the client chooses an initial sequence number (client_isn) and puts this number in the sequence number field of the initial TCP SYN segment. This segment is encapsulated within an IP datagram and sent to the server.

• Step 2. Once the IP datagram containing the TCP SYN segment arrives at the server host (assuming it does arrive!), the server extracts the TCP SYN segment from the datagram, allocates the TCP buffers and variables to the connection, and sends a connection-granted segment to the client TCP. This connection-granted segment also contains no application-layer data. However, it does contain three important pieces of information in the segment header. First, the SYN bit is set to 1. Second, the acknowledgment field of the TCP segment header is set to client_isn+l. Finally, the server chooses its own initial sequence number (server_isn) and puts this value in the sequence number field of the TCP segment header. This connection-granted segment is saying, in effect, "I received your SYN packet to start a connection with your initial sequence number, client_isn. I agree to establish this connection. My own initial sequence number is server_isn." The connection-granted segment is sometimes re-ferred to as a **SYNACK** segment.

• **Step 3.** Upon receiving the connection-granted segment, the client also allocates buffers and variables to the connection. The client host then sends the server yet another segment; this last segment acknowledges the server's connection-granted segment (the client does so by putting the value server_isn+l in the acknowledgment field of the TCP segment header). The SYN bit is set to 0, since the connection is established.

Once the preceding three steps have been completed, the client and server hosts can send segments containing data to each other. In each of these future segments, the SYN bit will be set to zero. Note that in order to establish the connection, three packets are sent between the two hosts, as illustrated in Figure 3-11. For this reason, this connection-establishment procedure is often referred to as a **three-way handshake.** Several aspects of the TCP three-way handshake are explored in the homework problems (Why are initial sequence numbers needed? Why is a three-way handshake, as opposed to a two-way handshake, needed?). It's interesting to note that a rock climber and a belayer (who is stationed below the rock climber and whose job it is to handle the climber's safety rope) use a three-way-handshake communication protocol that is identical to TCP's to ensure that both sides are ready before the climber begins ascent.

All good things must come to an end, and the same is true with a TCP connection. Either of the two processes participating in a TCP connection can end the connection. When a connection ends, the "resources" (that is, the buffers and variables) in the hosts are de-allocated. As an example, suppose the client decides to close the connection, as shown in Figure 3.12. The client application process issues a close command. This causes the client TCP to send a special TCP segment to the server process. This special segment has a flag bit in the segment's header, the FIN bit (see Figure 3.12), set to 1. When the server receives this segment, it sends the client an acknowledgment segment in return. The server then sends its own shutdown segment, which has the FIN bit set to 1. Finally, the client acknowledges the server's shutdown segment. At this point, all the resources in the two hosts are now de-allocated.

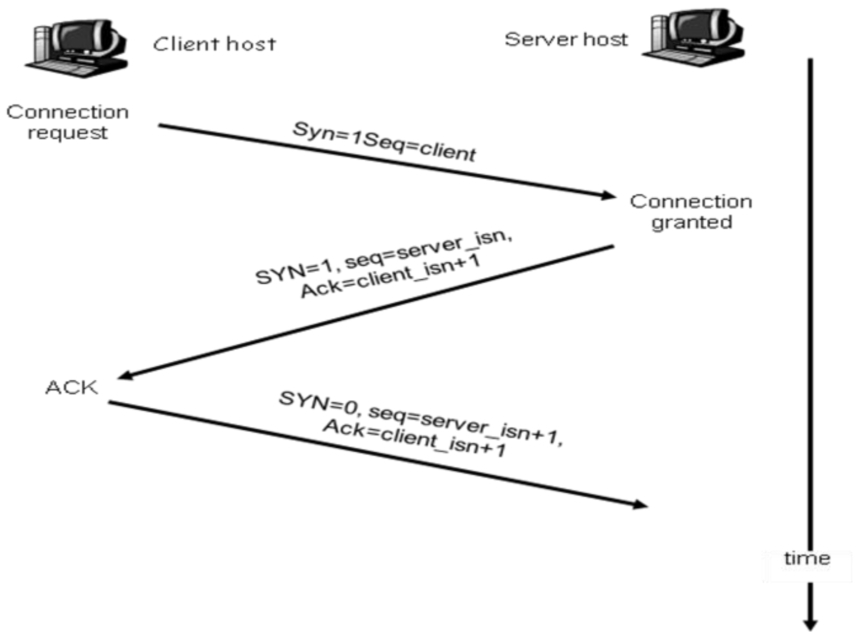

Client host
Server host
Connection request
Syn=1Seq=client
Connection granted
SYN=1, seq=server_isn, Ack=client_isn+1
ACK
SYN=0, seq=server_isn+1, Ack=client_isn+1
time

Figure 3-11 TCP three-way handshake: segment exchange

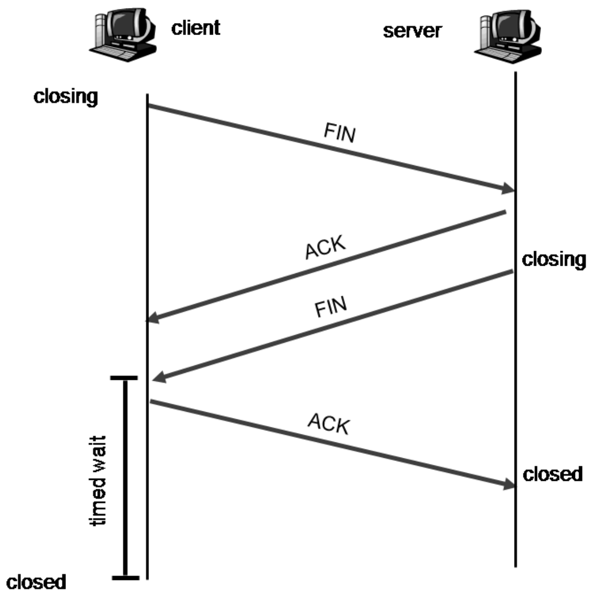


Figure 3-12 closing a TCP connection

During the life of a TCP connection, the TCP protocol running in each host makes transitions through various **TCP states.** Figure 3.13 illustrates a typical sequence of TCP states that are visited by the *client* TCP. The client TCP begins in the CLOSED state. The application on the client side initiates a new TCP connection. This causes TCP in the client to send a SYN segment to TCP in the server. After having sent the SYN segment, the client TCP enters the SYN_SENT state. While in the SYN_SENT state, the client TCP waits for a segment from the server TCP that includes an acknowledgment for the client's previous segment and has the SYN bit set to 1. Having received such a segment, the client TCP enters the ESTABLISHED state. While in the ESTABLISHED state, the TCP client can send and receive TCP segments containing payload (that is, application-generated) data.

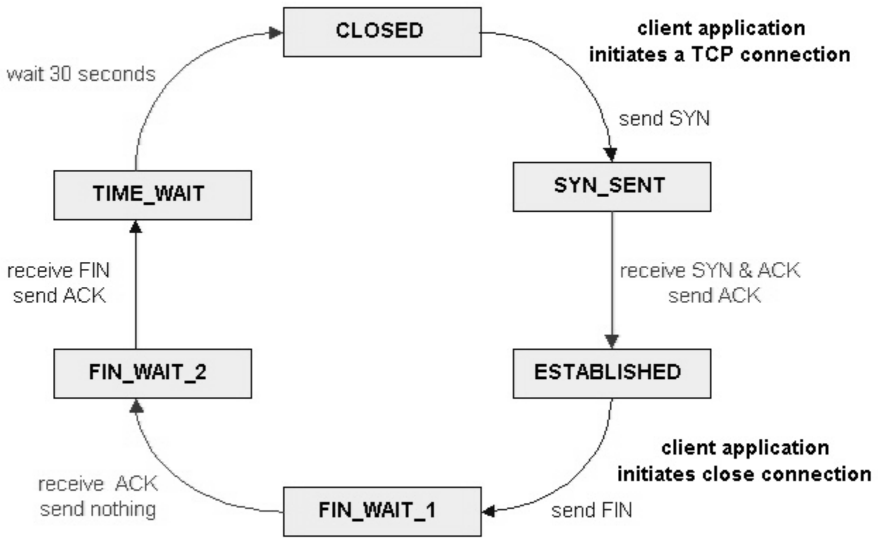


Figure 3-13 A typical sequence of TCP states visited by a client TCP

Suppose that the client application decides it wants to close the connection. (Note that client TCP to send a TCP segment with the FIN bit set to 1 and to enter the FIN_WAIT_1 state. While in the FIN_WAIT_1 state, the client TCP waits for a TCP segment from the server with an acknowledgment. When it receives this segment, the client TCP enters the FIN_WAIT_2 state. While in the FIN_WAIT_2 state, the client waits for another segment from the server with the FIN bit set to 1; after receiving this segment, the client TCP acknowledges the server's segment and enters the TIME_WAIT state. The TIME_WAIT state lets the TCP client resend the final acknowledgment in case the ACK is lost. The time spent in the TIME_WAIT state is implementation-dependent, but typical values are 30 seconds, 1 minute, and 2 minutes. After the wait, the connection formally closes and all resources on the client side (including port numbers) are released.

Figure 3.14 illustrates the series of states typically visited by the server-side TCP, assuming the client begins connection tear-down. The transitions are self-explanatory. In these two state-transition diagrams, we have only shown how a TCP connection is normally established and shut down. We have not described what happens in certain pathological scenarios, for example, when both sides of a connection want to shut down at the same time.

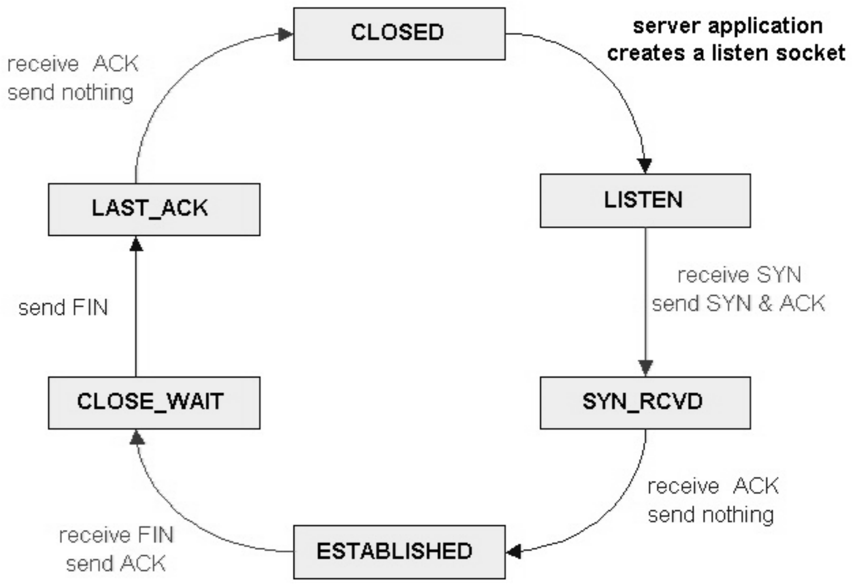


Figure 3-14 A Typical sequence of TCP states visited by a server-side TCP

Summary

We began this unit (lesson-1) by studying the services that a transport layer protocol can provide to network applications. At one extreme, the transport layer protocol can be very simple and offer a no-frills service to applications, providing only the multiplexing/ demultiplexing function for communicating processes. The Internet's UDP protocol is an example of such a no-frills (and no-thrills, from the perspective of someone interested in networking) transport-layer protocol. At the other extreme, a transport layer protocol can provide a variety of guarantees to applications, such as reliable delivery of data, delay guarantees and bandwidth guarantees. Nevertheless, the services that a transport protocol can provide are often constrained by the service model of the underlying network-layer protocol. If the network layer protocol cannot provide delay or bandwidth guarantees to transport-layer segments, then the transport layer protocol cannot provide delay or bandwidth guarantees for the messages sent between processes.

We learned in second lesson that a transport layer protocol can provide reliable data transfer even if the underlying network layer is unreliable. We saw that providing reliable data transfer has many subtle points, but that the task can be accomplished by carefully combining acknowledgments, timers, retransmissions and sequence numbers.

Although we covered reliable data transfer in this chapter, we should keep in mind that reliable data transfer can be provided by link, network, transport or application layer protocols. Any of upper four layers of the protocol stack can implement acknowledgments, timers, retransmissions and sequence numbers and provide reliable data transfer to the layer above. In fact, over the years, engineers and computer scientists have independently designed and implemented link, network, transport and application layer protocols that provide reliable data transfer (although many of these protocols have quietly disappeared).

In third lesson we took a close look at TCP, the Internet's connection-oriented and reliable transport-layer protocol. We learned that TCP is complex, involving connection management, flow control, round-trip time estimation, as well as reliable data transfer. In fact, TCP is actually more complex that we made it out to be -- we intentionally did not discuss a variety of TCP patches fixes, and improvements that are widely implemented in various versions of TCP. All of this complexity, however, is hidden from the network application. If a client on one host wants to reliably send data to a server on another host, it simply opens a TCP socket to the server and then pumps data into that socket. The client-server application is oblivious to all of TCP's complexity.

Exercise

1) Consider a TCP connection between host A and host B. Suppose that the TCP segments traveling from host A to host B have source port number x and destination port number y. What are the source and destination port numbers for the segments travelling from host B to host A?

2) Describe why an application developer may choose to run its application over UDP rather than TCP.

3) Is it possible for application to enjoy reliable data transfer even when the application runs over UDP? If so, how?

4) True or False:

a) Host A is sending host B a large file over a TCP connection. Assume host B has no data to send A. Host B will not send acknowledgements to host A because B cannot piggyback the acknowledgementson data?

b) The size of the TCP RcvWindow never changes throughout the duration of the connection?

c) Suppose host A is sending host B a large file over a TCP connection. The number of unacknowledged bytes that A sends cannot exceed the size of the receive buffer?

d) Suppose host A is sending a large file to host B over a TCP connection. If the sequence number for a segment of this connection is m, then the sequence number for the subsequent segment will necessarily be m+1?

e) The TCP segment has a field in its header for RcvWindow?

f) Suppose that the last SampleRTT in a TCP connection is equal to 1 sec. Then Timeout for the connection will necessarily be set to a value >= 1 sec.

g) Suppose host A sends host B one segment with sequence number 38 and 4 bytes of data. Then in this same segment the acknowledgement number is necessarily 42?

5) Suppose A sends two TCP segments back-to-back to B. The first segment has sequence number 90; the second has sequence number 110. a) How much data is the first segment? b) Suppose that the first segment is lost, but the second segment arrives at B. In the acknowledgement that B sends to A, what will be the acknowledgment number?

6) Consider the Telnet; a few seconds after the user types the letter 'C' the user types the letter 'R'. After typing the letter 'R' how many segments are sent and what is put in the sequence number and acknowledgement fields of the segments.

7) Suppose client A initiates an FTP session with server S. At about the same time, client B also initiates an FTP session with server S. Provide possible source and destination port numbers for:

(a) the segments sent from A to S?

(b) the segments sent from B to S?

(c) the segments sent from S to A?

(d) the segments sent from S to B?

(e) If A and B are different hosts, is it possible that the source port numbers in the segments from A to S are the same as those from B to S?

(f) How about if they are the same host?

8) UDP and TCP use 1's complement for their checksums. Suppose you have the following three 8-bit words: 01010101, 01110000, and 11001100. What is the 1's complement of the sum of these words? Show all work.

Why it is that UDP take the 1's complement of the sum, i.e., why not just use the sum? With the 1's complement scheme, how does the receiver detect errors? Is it possible that a 1-bit error will go undetected? How about a 2-bit error?

9) Protocol rdt2.1 uses both ACK's and NAKs. Redesign the protocol, adding whatever additional protocol mechanisms are needed, for the case that only ACK messages are used. Assume that packets can be corrupted, but not lost. Give the sender and receiver FSMs, and a trace of your protocol in operation. Show also how the protocol works in the case of no errors, and show how your protocol recovers from channel bit errors.

10) In protocol rdt3.0, the ACK packets flowing from the receiver to the sender do not have sequence numbers (although they do have an ACK field that contains the sequence number of the packet they are acknowledging). Why is it that our ACK packets do not require sequence numbers?

11) Draw the FSM for the receiver side of protocol rdt 3.0.

12) Consider a channel that can lose packets but has a maximum delay that is known. Modify protocol rdt2.1 to include sender timeout and retransmit. Informally argue why your protocol can communicate correctly over this channel.

13) Design a reliable, pipelined, data transfer protocol that uses only negative acknowledgements. How quickly will your protocol respond to lost packets when the arrival rate of data to the sender is low? Is high?

14) Consider transferring an enormous file of L bytes from host A to host B. Assume an MSS of 1460 bytes.

a) What us the maximum length of L such that TCP sequence numbers is not exhausted? Recall that the TCP number field has four bytes.

b) For the L you obtain in (a), find how long it takes to transmit the file. Assume that a total of 66 bytes of transport, network and data-link header are added to each segment before the resulting packet is sent out over a 10 Mbps link. Ignore flow control and congestion control, so A can pump out the segments back-to-back and continuously.

15) Suppose TCP increased its congestion window by two rather than by one for each received acknowledgement during slow start. Thus the first window consists of one segment, the second of three segments, the third of nine segments, etc. For this slow-start procedure:

a) Express K in terms of O and S. b) Express Q in terms of RTT, S and R.

c) Express latency in terms of P = min (K-1, Q), O, R and RTT.

UNIT – III

NETWORK LAYER AND ROUTING

1. Introduction and Network services

Structure

Objectives

- Define basic functions of the network layer;
- Explain concept of network service models
- Explain the concept of routing;
- Explain the concept of datagram's switching, and
- Virtual switching network.

1.1 Introduction

We saw in the previous chapter that the transport layer provides communication service between *two processes* running on two different hosts. In order to provide this service, the transport layer relies on the services of the network layer, which provides a communication service between *hosts*. In particular, the network-layer moves transport-layer segments from one host to another. At the sending host, the transport layer segment is passed to the network layer. The network layer then "somehow" gets the segment to the destination host and passes the segment up the protocol stack to the transport layer. Exactly how the network layer moves a segment from the transport layer of an origin host to the transport layer of the destination host is the subject of this chapter. We will see that unlike the transport layers, the network layer *requires the coordination of each and every host and router in the network.* Because of this, network layer protocols are among the most challenging (and therefore interesting!) in the protocol stack.

NOTES

1-2 Network layer Functions

Figure 1-1 shows a simple network with two hosts (H1 and H2) and four routers (R1, R2, R3 and R4). The role of the network layer in a sending host is to begin the packet on its journey to the receiving host. For example, if H1 is sending to H2, the network layer in host H1 transfers these packets to it nearby router, R2. At the receiving host (e.g., H2), the network layer receives the packet from its nearby router (in this case, R3) and delivers the packet up to the transport layer at H2. The primary role of the routers is to "switch" packets from input links to output links. Note that the routers in Figure 1-1 are shown with a truncated protocol stack, i.e., with no upper layers above the network layer, since routers do not run transport and application layer protocols such as those we examined in unit-1 and unit-2.

The role of the network layer is thus deceptively simple to transport packets from a sending host to a receiving host. To do so, three important network layer functions can be identified:

- **Path Determination.** The network layer must determine the route or path taken by packets as they flow from a sender to a receiver. The algorithms that calculate these paths are referred to as **routing algorithms**. A routing algorithm would determine, for example, whether packets from H1 to H2 flow along the path R2-R1-R3 or path R2-R4-R3 (or any other path between H1 and H2). Much of this chapter will focus on routing algorithms. In Section 4.2 we will study the theory of routing algorithms, concentrating on the two most prevalent classes of routing algorithms: link state routing and distance vector routing. We will see that the complexity of routing algorithms grows considerably as the number of routers in the network increases.

- **Switching.** When a packet arrives at the input to a router, the router must move it to the appropriate output link. For example, a packet arriving from host H1 to router R2 must either be forwarded towards H2 either along the link from R2 to R1 or along the link from R2 to R4.

- **Call Setup.** Recall that in our study of TCP, a three-way handshake was required before data actually flowed from sender to receiver. This allowed the sender and receiver to setup the needed state information (e.g., sequence number and initial flow control window size). In an analogous manner, some network layer architectures (e.g., ATM) requires that the routers along the chosen path from source to destination handshake with each other in order to setup state before data actually begins to flow. In the network layer, this process is referred to as **call setup**. The network layer of the Internet architecture does not perform any such call setup.

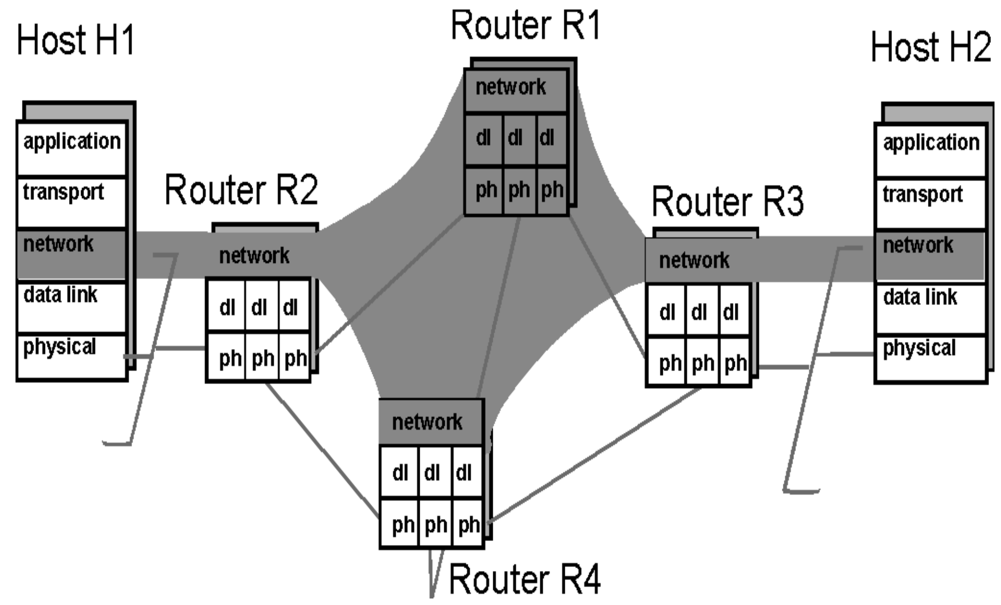


Figure 1-1 The Network layer

Before delving into the details of the theory and implementation of the network layer, however, let us first take the broader view and consider what different types of *service* might be offered by the network layer.

1-3 Network Service Model

When the transport layer at a sending host transmits a packet into the network (i.e., passes it down to the network layer at the sending host), can the transport layer count on the network layer to deliver the packet to the destination? When multiple packets are sent, will they be delivered to the transport layer in the receiving host in the order in which they were sent? Will the amount of time between the sending of two sequential packet transmissions be the same as the amount of time between their reception? Will the network provide any feedback about congestion in the network? What is the abstract view (properties) of the channel connecting the transport layer in the two hosts? The answers to these questions and others are determined by the *service model* provided by the network layer. The **network service model** defines the characteristics of end-to-end transport of data between one "edge" of the network and the other, i.e., between sending and receiving end systems.

1-3-1 Datagram or Virtual Circuit?

Perhaps the most important abstraction provided by the network layer to the upper layers is whether or not the network layer uses **virtual circuits (VCs)** or not. You may recall from Chapter 1 that a virtual-circuit packet network behaves much like a telephone network, which uses "real circuits" as opposed to "virtual circuits". There are three identifiable phases in a virtual circuit:

- **VC setup.** During the setup phase, the sender contacts the network layer, specifies the receiver address, and waits for the network to setup the VC. The network layer determines the path between sender and receiver, i.e., the series of links and switches through which all packets of the VC will travel. As discussed in Chapter 1, this typically involves updating tables in each of the packet switches in the path. During VC setup, the network layer may also reserve resources (e.g., bandwidth) along the path of the VC.
- **Data transfer.** Once the VC has been established, data can begin to flow along the VC.
- **Virtual circuit teardown.** This is initiated when the sender (or receiver) informs the network layer of its desire to terminate the VC. The network layer will then typically inform the end system on the other side of the network of the call termination, and update the tables in each of the packet switches on the path to indicate that the VC no longer exists.

There is a subtle but important distinction between VC setup at the network layer and connection setup at the transport layer (e.g., the TCP 3-way handshake we studied in unit 2). Connection setup at the transport layer only involves the two end systems. The two end systems agree to communicate and together determine the parameters (e.g., initial sequence number, flow control window size) of their transport level connection before data actually begins to flow on the transport level connection. Although the two end systems are aware of the transport-layer connection, the switches within the network are completely oblivious to it. On the other hand, with a virtual-circuit network layer, *packet switches are involved in virtual-circuit setup, and each packet switch is fully aware of all the VCs passing through it.*

The messages that the end systems send to the network to indicate the initiation or termination of a VC, and the messages passed between the switches to set up the VC (i.e. to modify switch tables) are known as **signaling messages** and the protocols used to exchange these messages are often referred to as **signaling protocols.** VC setup is shown pictorially in Figure 1-2.

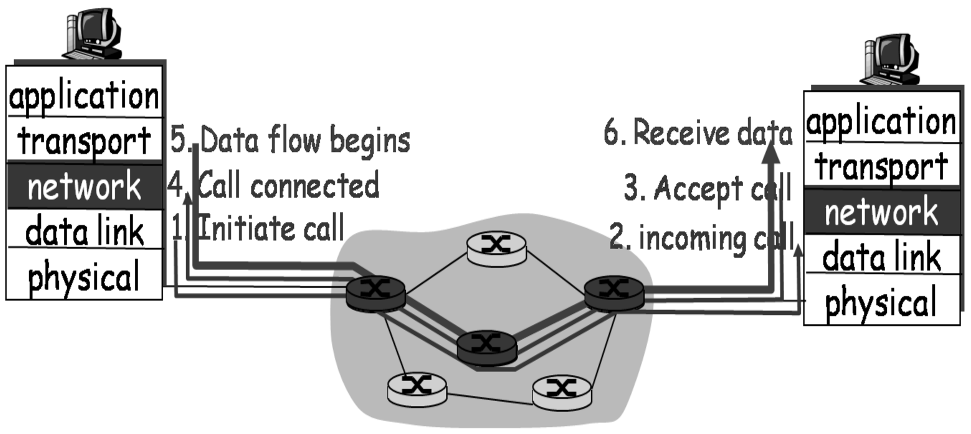


Figure 1-2 Virtual-circuit service model

With a **datagram network layer**, each time an end system wants to send a packet; it stamps the packet with the address of the destination end system, and then pops the packet into the network. As shown in Figure 1-3, this is done without any VC setup. Packet switches (called "routers" in the Internet) do not maintain any state information about VCs because there are no VCs! Instead, packet switches route a packet towards its destination by examining the packet's destination address, indexing a routing table with the destination address, and forwarding the packet in the direction of the destination. Because routing tables can be modified at any time, a series of packets sent from one end system to another may follow different paths through the network and may arrive out of order. The Internet uses a datagram network layer.

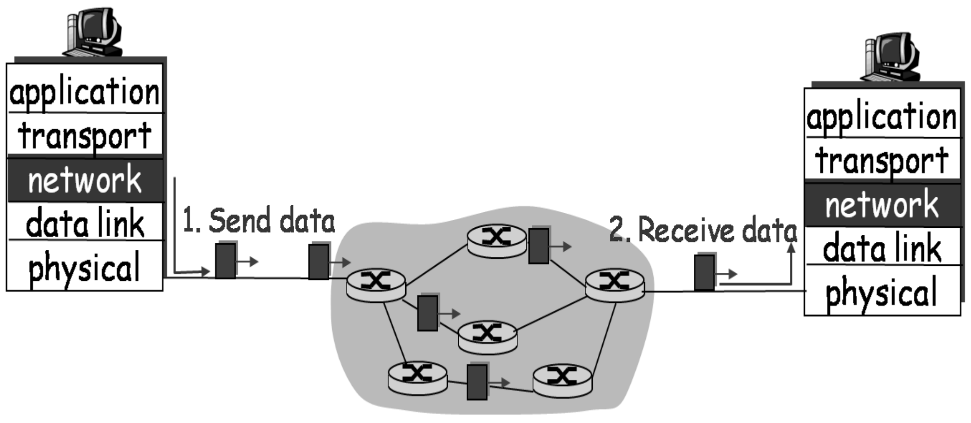


Figure 1-3 Datagram service model

You may recall from unit-1 that a packet-switched network typically offers either a VC service or a datagram service to the transport layer, and not both services. For example, an ATM network offers only a VC service to the ATM transport layer (more precisely, to the ATM adaptation layer), and the Internet offers only a datagram service to the transport layer. The transport layer in turn offers services to communicating processes at the application layer. For example, TCP/IP networks (such as the Internet) offer a connection-oriented service (using TCP) and connectionless service (UDP) to its communicating processes.

An alternative terminology for VC service and datagram service is **network-layer connection-oriented service** and **network-layer connectionless service,** respectively. Indeed, the VC service is a sort of connection-oriented service, as it involves setting up and tearing down a connection-like entity, and maintaining connection state information in the packet switches. The datagram service is a sort of connectionless service in that it doesn't employ connection-like entities. Both sets of terminology have advantages and disadvantages, and both sets are commonly used in the networking literature. We decided to use in this book the "VC service" and "datagram service" terminology for the network layer, and reserve the "connection-oriented service" and "connectionless service" terminology for the transport layer. We believe this decision will be useful in helping the reader delineate the services offered by the two layers.

1-3-2 Internet and ATM Network Service Models

The key aspects of the service model of the Internet and ATM network architectures are summarized in Table 1-1. We do not want to delve deeply into the details of the service models here (it can be quite "dry" and detailed discussions can be found in the standards themselves. A comparison between the Internet and ATM service models is, however, quite instructive.

Network Architecture	Service Model	Bandwidth Guarantee	No Loss Guarantee	Ordering	Timing	Congestion indication
Internet	Best Effort	None	None	Any order possible	Not maintained	None
ATM	CBR	Guaranteed constant rate	Yes	In order	maintained	congestion will not occur
ATM	VBR	Guaranteed rate	Yes	In order	maintained	congestion will not occur
ATM	ABR	Guaranteed minimum	None	In order	Not maintained	Congestion indication provided
ATM	UBR	None	None	In order	Not maintained	None

Table 1-1: Internet and ATM Network Service Models

The current Internet architecture provides only one service model, the datagram service, which is also known as "**best effort service**." From Table 1-1, it might appear that best effort service is a euphemism for "no service at all." With best effort service, timing between packets is not guaranteed to be preserved, packets are not guaranteed to be received in the order in which they were sent, nor is the eventual delivery of transmitted packets guaranteed. Given this definition, a network which delivered *no* packets to the destination would satisfy the definition best effort delivery service (Indeed, today's congested public Internet might sometimes appear to be an example of a network that does so!). As we will discuss shortly, however, there are sound reasons for such a minimalist network service model. The Internet's best-effort only service model is currently being extended to include so-called "integrated services" and "differentiated service."

Constant bit rate (CBR) network service was the first ATM service model to be standardized, probably reflecting the fact that telephone companies were the early prime movers behind ATM, and CBR network service is ideally suited for carrying real-time, constant-bit-rate, streamline audio (e.g., a digitized telephone call) and video traffic. The goal of CBR service is conceptually simple to make the network connection look like a dedicated copper or fiber connection between the sender and receiver. With CBR service, ATM cells are carried across the network in such a way that the end-end delay experienced by a cell (the so-called cell transfer delay, CDT), the variability in the end-end delay (often referred to as "jitter" or "cell delay variation, CDV)"), and the fraction of cells that are lost or deliver late (the so-called cell loss rate, CLR) are guaranteed to be less than some specified values. Also, an allocated transmission rate (the peak cell rate, PCR) is defined for the connection and the sender is expected to offer data to the network at this rate. The values for the PCR, CDT, CDV, and CLR are agreed upon by the sending host and the ATM network when the CBR connection is first established.

A second conceptually simple ATM service class is **Unspecified Bit Rate (UBR) network service**. Unlike CBR service, which guarantees rate, delay, delay jitter, and loss, UBR makes no guarantees at all other than in-order delivery of cells (that is, cells that are fortunate enough to make it to the receiver). With the exception of in-order delivery, UBR service is thus equivalent to the Internet best effort service model. As with the Internet best effort service model, UBR also provides no feedback to the sender about whether or not a cell is dropped within the network. For reliable transmission of data over a UBR network, higher layer protocols (such as those we studied in the previous chapter) are needed. UBR service might be well suited for non-interactive data transfer applications such as email and newsgroups.

If UBR can be thought of as a "best effort" service, then **Available Bit Rate (ABR) network service** might best be characterized as a "better" best effort service model. The two most important additional features of ABR service over UBR service are:

- A minimum cell transmission rate (MCR) is guaranteed to a connection using ABR service. If, however, the network has enough free resources at a given time, a sender may actually be able to successfully send traffic at a *higher* rate than the MCR.

- Congestion feedback from the network. An ATM network provides feedback to the sender (in terms of a congestion notification bit, or a lower rate at which to send) that controls how the sender should adjust its rate between the MCR and some peak cell rate (PCR). ABR senders must decrease their transmission rates in accordance with such feedback.

ABR provides a minimum bandwidth guarantee, but on the other hand will attempt to transfer data as fast as possible (up to the limit imposed by the PCR). As such, ABR is well suited for data transfer where it is desirable to keep the transfer delays low (e.g., Web browsing).

The final ATM service model is **Variable Bit Rate (VBR) network service**. VBR service comes in two flavors (and in the ITU specification of VBR-like service comes in *four* flavors -- perhaps indicating a service class with an identity crisis!). In real-time VBR service, the acceptable cell loss rate, delay, and delay jitter are specified as in CBR service. However, the actual source rate is allowed to vary according to parameters specified by the user to the network. The declared variability in rate may be used by the network (internally) to more efficiently allocate resources to its connections, but in terms of the loss, delay and jitter seen by the sender, the service is essentially the same as CBR service. While early efforts in defining a VBR service models were clearly targeted towards real-time services (e.g., as evidenced by the PCR, CDT, CDV and CLR parameters), a second flavor of VBR service is now targeted towards non-real-time services and provides a cell loss rate guarantee. An obvious question with VBR is what advantages it offers over CBR (for real-time applications) and over UBR and ABR for non-real-time applications. Currently, there is not enough (any?) experience with VBR service to answer this questions.

1-4 Origins of Datagram and Virtual Circuit Service

The evolution of the Internet and ATM network service models reflects their origins. With the notion of a virtual circuit as a central organizing principle, and an early focus on CBR services, ATM reflects its roots in the telephony world (which uses "real circuits"). The subsequent definition of UBR and ABR service classes acknowledges the importance of the types of data applications developed in the data networking community. Given the VC architecture and a focus on supporting real-time traffic with *guarantees* about the level of received performance (even with data-oriented services such as ABR), the network layer is *significantly more complex* than the best effort Internet. This too, is in keeping with the ATM's telephony heritage. Telephone networks, by necessity, had their "complexity' within the network, since they were connecting "dumb" end-system devices such as a rotary telephone (For those too young to know, a rotary phone is a non-digital telephone with no buttons only a dial).

The Internet, on the other hand, grew out of the need to connect computers (i.e., more sophisticated end devices) together. With sophisticated end-systems devices, the Internet architects chose to make the network service model (best effort) as simple as possible and to implement any additional functionality (e.g., reliable data transfer), as well as any new application level network services at a higher layer, at the end systems. This inverts the model of the telephone network, with some interesting consequences:

- The resulting network service model which made minimal (no!) service guarantees (and hence posed minimal requirements on the network layer) also made it easier to *interconnect* networks that used very different link layer technologies (e.g., satellite, Ethernet, fiber, or radio) which had very different characteristics (transmission rates, loss characteristics).

2. Routing Principles

Structure

Objectives

- Understand of routing principles;
- Understand the classification of routing algorithms;
- Understand how the Link state routing algorithm works;
- Understand how the distance vector routing algorithm works;
- Comparison of Link State and Distance Vector Routing Algorithms
- Discuss other routing algorithms

2.1 Introduction

"Rö´ ting" is what fans do at a football game, what pigs do for truffles under oak trees in the Vaucluse, and what nurserymen intent on propagation do to cuttings from plants. "Rou´ ting" is how one creates a beveled edge on a table top, or sends a corps of infantrymen into full-scale, disorganized retreat. Either pronunciation is correct for "routing," which refers to the process of discovering, selecting, and employing paths from one place to another (or to many others) in a network. The British prefer the spelling *routeing,* presumably to distinguish what happens in networks from what happened to the British in New Orleans in 1814. Since the *Oxford English Dictionary* is much heavier than any dictionary of American English, British English generally prevails in the documents produced by ISO and CCITT; wherefore, most of the international standards for routing protocols use the *routeing* spelling. Since this spelling would be unfamiliar to many readers.

A simple definition of routing is "learning how to get from here to there." In some cases, the term *routing* is used in a very strict sense to refer *only* to the process of obtaining and distributing information ("learning"), but not to the process of using that information to actually get from one place to another (for which a different term, *forwarding,* is reserved). Since it is difficult to grasp the usefulness of information that is acquired but never used, we employ the term *routing* to refer in general to all the things that are done to discover and advertise paths from here to there and to actually move packets from here to there when necessary. The distinction between routing and forwarding is preserved in the formal discussion of the functions performed by OSI end systems and intermediate systems, in which context the distinction is meaningful.

2-2 Routing Principles

In order to transfer packets from a sending host to the destination host, the network layer must determine the *path* or *route* that the packets are to follow. Whether the network layer provides a datagram service (in which case different packets between a given host-destination pair may take different routes) or a virtual circuit service (in which case all packets between a given source and destination will take the same path), the network layer must nonetheless determine the path for a packet. This is the job of the network layer **routing protocol.**

At the heart of any routing protocol is the algorithm (the "routing algorithm") that determines the path for a packet. The purpose of a routing algorithm is simple: given a set of routers, with links connecting the routers, a routing algorithm finds a "good" path from source to destination. Typically, a "good" path is one which has "least cost," but we will see that in practice, "real-world" concerns such as policy issues (e.g., a rule such as "router X, belonging to organization Y should not forward any packets originating from the network owned by organization Z") also come into play to complicate the conceptually simple and elegant algorithms whose theory underlies the practice of routing in today's networks.

The graph abstraction used to formulate routing algorithms is shown in Figure 2-1. (To view some graphs representing real network maps. Here, nodes in the graph represent routers - the points at which packet routing decisions are made - and the lines ("edges" in graph theory terminology) connecting these nodes represent the physical links between these routers. A link also has a value representing the "cost" of sending a packet across the link. The cost may reflect the level of congestion on that link (e.g., the current average delay for a packet across that link) or the physical distance traversed by that link (e.g., a transoceanic link might have a higher cost than a terrestrial link). For our current purposes, we will simply take the link costs as a given and won't worry about how they are determined.

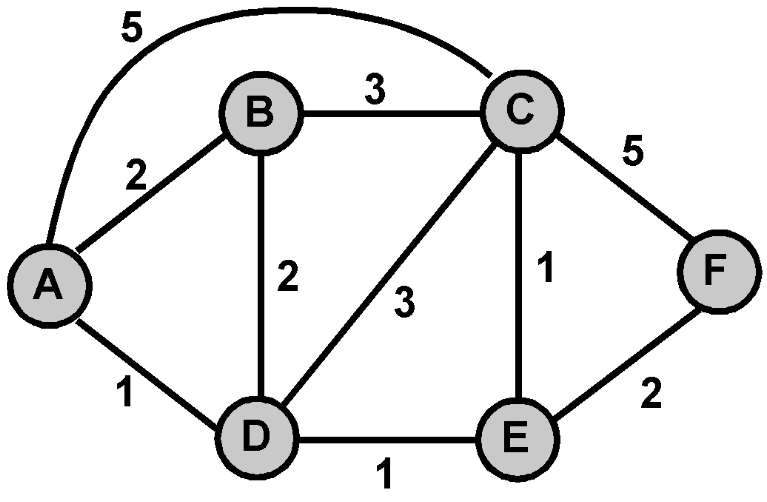


Figure 2-1: Abstract model of a network

Given the graph abstraction, the problem of finding the least cost path from a source to a destination requires identifying a series of links such that:

- the first link in the path is connected to the source
- the last link in the path is connected to the destination
- for all *i,* the *i* and *i-1*st link in the path are connected to the same node
- for the **least cost path**, the sum of the cost of the links on the path is the minimum over all possible paths between the source and destination. Note that if all link costs are the same, the least cost path is also the **shortest path** (i.e., the path crossing the smallest number of links between the source and the destination).

In Figure 2-1, for example, the least cost path between nodes A (source) and C (destination) is along the path ADEC. (We will find it notationally easier to refer to the path in terms of the nodes on the path, rather than the links on the path).

2-3 Classification of Routing Algorithms

As a simple exercise, try finding the least cost path from nodes A to F, and reflect for a moment on how you calculated that path. If you are like most people, you found the path from A to F by examining Figure 2-1, tracing a few routes from A to F, and somehow convincing yourself that the path you had chosen was the least cost among all possible paths (Did you check all of the 12 possible paths between A and F? Probably not!). Such a calculation is an example of a centralized routing algorithm. Broadly, one way in which we can classify routing algorithms is according to whether they are centralized or decentralized:

- A **global routing algorithm** computes the least cost path between a source and destination using complete, global knowledge about the network. That is, the algorithm takes the connectivity between all nodes and all links costs as inputs. This then requires that the algorithm somehow obtain this information before actually performing the calculation. The calculation itself can be run at one site (a centralized global routing algorithm) or replicated at multiple sites. The key distinguishing feature here, however, is that a global algorithm has *complete* information about connectivity and link costs. In practice, algorithms with global state information are often referred to as **link state algorithms**, since the algorithm must be aware of the state (cost) of each link in the network. We will study a global link state algorithm in lesson-1(unit-3).

- In a **decentralized routing algorithm,** the calculation of the least cost path is carried out in an iterative, distributed manner. No node has complete information about the costs of all network links. Instead, each node begins with only knowledge of the costs of its own directly attached links and then through an iterative process of calculation and exchange of information with its neighboring nodes (i.e., nodes which are at the "other end" of links to which it itself is attached) gradually calculates the least cost path to a destination, or set of destinations. We will study a decentralized routing algorithm known as a **distance vector algorithm.** It is called a distance vector algorithm because a node never actually knows a complete path from source to destination. Instead, it only knows the direction (which neighbor) to which it should forward a packet in order to reach a given destination along the least cost path, and the cost of that path from itself to the destination.

A second broad way to classify routing algorithms is according to whether they are **static** or **dynamic.** In static routing algorithms, routes change very slowly over time, often as a result of human intervention (e.g., a human manually editing a router's forwarding table). Dynamic routing algorithms change the routing paths as the network traffic loads (and the resulting delays experienced by traffic) or topology change. A dynamic algorithm can be run either periodically or in direct response to topology or link cost changes. While dynamic algorithms are more responsive to network changes, they are also more susceptible to problems such as routing loops and oscillation in routes.

Only two types of routing algorithms are typically used in the Internet: a dynamic global link state algorithm, and a dynamic decentralized distance vector algorithm.

2-4 A Link State Routing Algorithm

Recall that in a link state algorithm, the network topology and all link costs are known, i.e., available as input to the link state algorithm. In practice this is accomplished by having each node **broadcast** the identities and costs of its attached links to *all* other routers in the network. This **link**

state broadcast can be accomplished without the nodes having to initially know the identities of all other nodes in the network A node need only know the identities and costs to its directly-attached neighbors; it will then learn about the topology of the rest of the network by receiving link state broadcast from other nodes. (In Chapter 5, we will learn how a router learns the identities of its directly attached neighbors). The result of the nodes' link state broadcast is that all nodes have an identical and complete view of the network. Each node can then run the link state algorithm and compute the same set of least cost paths as every other node.

The link state algorithm we present below is known as Dijkstra's algorithm, named after its inventor (a closely related algorithm is Prim's algorithm; for a general discussion of graph algorithms). It computes the least cost path from one node (the source, which we will refer to as A) to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the kth iteration of the algorithm, the least cost paths are known to k destination nodes, and among the least cost paths to all destination nodes, these k path will have the k smallest costs. Let us define the following notation:

- c(i,j): link cost from node i to node j. If nodes i and j are not directly connected, then c(i,j) = infty=α . We will assume for simplicity that c(i,j) equals c(j,i).
- D(v): the cost of path from the source node to destination v that has currently (as of this iteration of the algorithm) the least cost.
- p(v): previous node (neighbor of v) along current least cost path from source to v
- N: set of nodes whose shortest path from the source is definitively known

The link state algorithm consists of an initialization step followed by a loop. The number of times the loop is executed is equal to the number of nodes in the network. Upon termination, the algorithm will have calculated the shortest paths from the source node to every other node in the network.

2-3-1 ink State (LS) Algorithm:

```
Initialization:
  N = {A}
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v)
      else D(v) = infty

 Loop
   find w not in N such that D(w) is a minimum
   add w to N
   update D(v) for all v adjacent to w and not in N:
      D(v) = min( D(v), D(w) + c(w,v) )
   /* new cost to v is either old cost to v or known
```

```
14     shortest path cost to w plus cost from w to v */
15  until all nodes in N
```

As an example, let us consider the network in Figure 2-1 and compute the shortest path from A to all possible destinations. A tabular summary of the algorithm's computation is shown in Table 2-1, where each line in the table gives the values of the algorithms variables at the end of the iteration. Let us consider the few first steps in detail:

step	N	D(B),p(B)	D(C),P(C)	D(D),P(D)	D(E),P(E)	D(F),p(F)
0	A	2,A	5,A	1,A	infty	infty
1	AD	2,A	4,D		2,D	infty
2	ADE	2,A	3,E			4,E
3	ADEB		3E			4E
4	ADEBC					4E
5	ADEBCF					

Table 2-1: Steps in running the link state algorithm on network in Figure 2-1

- **In the initialization step,** the currently known least path costs from A to its directly attached neighbors, B, C and D are initialized to 2, 5 and 1 respectively. Note in particular that the cost to C is set to 5 (even though we will soon see that a lesser cost path does indeed exist) since this is cost of the direct (one hop) link from A to C. The costs to E and F are set to infinity since they are not directly connected to A.
- **In the first iteration,** we look among those nodes not yet added to the set N and find that node with the least cost as of the end of the previous iteration. That node is D, with a cost of 1, and thus D is added to the set N. Line 12 of the LS algorithm is then performed to update D(v) for all nodes v, yielding the results shown in the second line (step 1) in Table 4.2-1. The cost of the path to B is unchanged. The cost of the path to C (which was 5 at the end of the initialization) through node D is found to have a cost of 4. Hence this lower cost path is selected and C's predecessor along the shortest path from A is set to D. Similarly, the cost to E (through D) is computed to be 2, and the table is updated accordingly.
- **In the second iteration**, nodes B and E are found to have the shortest path costs (2), and we break the tie arbitrarily and add E to the set N so that N now contains A, D, and E. The cost to the remaining nodes not yet in N, i.e., nodes B, C and F, are updated via line 12 of the LS algorithm , yielding the results shown in the third row in the above table.
- and so on ...

When the LS algorithm terminates, we have for each node, its predecessor along the least cost path from the source node. For each predecessor, we also have *its* predecessor and so in this manner we can construct the entire path from the source to all destinations.

What is the computation complexity of this algorithm? That is, given *n* nodes (not counting the source), how much computation must be done in the worst case to find the least cost paths from the source to all destinations? In the first iteration, we need to search through all *n* nodes to determine the node, *w,* not in N that has the minimum cost. In the second iteration, we need to check *n-1* nodes to determine the minimum cost; in the third iteration *n-2* nodes and so on. Overall, the total number of nodes we need to search through over all the iterations is n*(n+1)/2, and thus we say that the above implementation of the link state algorithm has worst case complexity of order *n* squared: $O(n^2)$. (A more sophisticated implementation of this algorithm, using a data structure known as a heap, can find the minimum in line 9 in logarithmic rather than linear time, thus reducing the complexity).

Before completing our discussion of the LS algorithm, let us consider a pathology that can arise with the use of link state routing. Figure 2-2 shows a simple network topology where link costs are equal to the load carried on the link, e.g., reflecting the delay that would be experienced. In this example, link costs are not symmetric, i.e., c (A,B) equals c(B,A) only if the load carried on both directions on the AB link is the same. In this example, node D originates a unit of traffic destined for A, node B also originates a unit of traffic destined for A, and node C injects an amount of traffic equal to e, also destined for A. The initial routing is shown in Figure 2-2a, with the link costs corresponding to the amount of traffic carried.

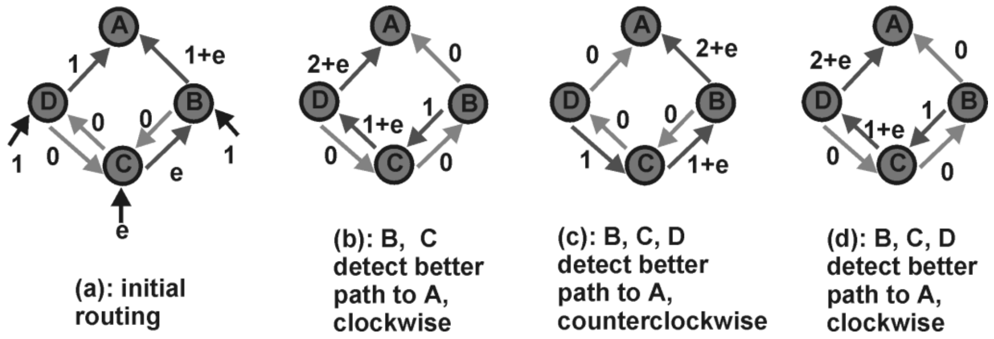

(a): initial routing
(b): B, C detect better path to A, clockwise
(c): B, C, D detect better path to A, counterclockwise
(d): B, C, D detect better path to A, clockwise
1
1+e
0
0
1
0
e
1
e
2+e
0
1+e
1
0
0
0
2+e
0
0
1
1+e
2+e
0
1+e
1
0
0

Figure 2-2: Oscillations with Link State routing

When the LS algorithm is next run, node C determines (based on the link costs shown in Figure 2-2a) that the clockwise path to A has a cost of 1, while the counterclockwise path to A (which it had been using) has a cost of 1+e. Hence C's least cost path to A is now clockwise. Similarly, B determines that its new least cost path to A is also clockwise, resulting in the routing and resulting path costs shown in Figure 2-2b. When the LS algorithm is run next, nodes B, C and D all detect that a zero cost path to A in the counterclockwise direction and all route their traffic to the counterclockwise routes. The next time the LS algorithm is run, B, C, and D all then route their traffic to the clockwise routes.

What can be done to prevent such oscillations in the LS algorithm? One solution would be to mandate that link costs not depend on the amount of traffic carried -- an unacceptable solution since one goal of routing is to avoid highly congested (e.g., high delay) links. Another solution is to insure that all routers do not run the LS algorithm at the same time. This seems a more reasonable solution, since we would hope that even if routers run the LS algorithm with the same periodicity, the execution instants of the algorithm would not be the same at each node. Interestingly, researchers have recently noted that routers in the Internet can self-synchronize among themselves, i.e., even though they initially execute the algorithm with the same period but at different instants of time, the algorithm execution instants can eventually become, and remain, synchronized at the routers. One way to avoid such self-synchronization is to purposefully introduce randomization into the period between execution instants of the algorithm at each node.

Having now studied the link state algorithm, let's next consider the other major routing algorithm that is used in practice today - the distance vector routing algorithm.

2-5 A Distance Vector Routing Algorithm

While the LS algorithm is an algorithm using global information, the **distance vector (DV)** algorithm is iterative, *asynchronous, and distributed.* It is distributed in that each node receives some information from one or more of its *directly* attached neighbors, performs a calculation, and may then distribute the results of its calculation back to its neighbors. It is iterative in that this process continues on until no more information is exchanged between neighbors. (Interestingly, we will see that the algorithm is self terminating there is no "signal" that the computation should stop; it just stops). The algorithm is asynchronous in that it does not require all of the nodes to operate in lock step with each other. We'll see that an asynchronous, iterative, self terminating, distributed algorithm is much more "interesting" and "fun" than a centralized algorithm.

The principal data structure in the DV algorithm is the **distance table** maintained at each node. Each node's distance table has a row for each destination in the network and a column for each of its directly attached neighbors. Consider a node X that is interested in routing to destination Y via its directly attached neighbor Z. Node X's **distance table entry,** $D^x(Y,Z)$ is the sum of the cost of the direct one hop link between X and Z, c(X,Z), plus neighbor Z's currently known minimum cost path from itself (Z) to Y. That is:

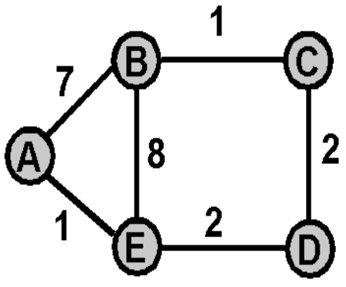
$$D^x(Y,Z) = c(X,Z) + \min_w\{D^z(Y,w)\} \qquad (2\text{-}1)$$

The $\min_w$ term in equation 2-1 is taken over all of Z's directly attached neighbors (including X, as we shall soon see).

Equation 2-1 suggests the form of the neighbor-to-neighbor communication that will take place in the DV algorithm -- each node must know the cost of each of its neighbors minimum cost path to each destination Thus, whenever a node computes a new minimum cost to some destination, it must inform its neighbors of this new minimum cost.

Before presenting the DV algorithm, let's consider an example that will help clarify the meaning of entries in the distance table. Consider the network topology and the distance table shown for node E in Figure 2-3. This is the distance table in node E once the Dv algorithm has converged. Let's first look at the row for destination A.

- Clearly the cost to get to A from E via the direct connection to A has a cost of 1. Hence $D^E(A,A) = 1$.
- Let's now consider the value of $D^E(A,D)$ - the cost to get from E to A, given that the first step along the path is D. In this case, the distance table entry is the cost to get from E to D (a cost of 2) plus whatever the minimum cost it is to get from D to A. Note that the minimum cost from D to A is 3 a path that passes right back through E! Nonetheless, we record the fact that the minimum cost from E to A given that the first step is via D has a cost of 5. We're left, though, with an uneasy feeling that the fact the path from E via D loops back through E may be the source of problems down the road (it will!).
- Similarly, we find that the distance table entry via neighbor B is $D^E(A,B) = 14$. Note that the cost is *not* 15. (Why?)



		cost to destination via		
	$D^E()$	A	B	D
destinat.	A	①	14	5
	B	7	8	⑤
	C	6	9	④
	D	4	11	②

Figure 2-3: A distance table example

A circled entry in the distance table gives the cost of the least cost path to the corresponding destination (row). The column with the circled entry identifies the next node along the least cost path to the destination. Thus, a node's **routing table** (which indicates which outgoing link should be used to forward packets to a given destination) is easily constructed from the node's distance table.

In discussing the distance table entries for node E above, we informally took a global view, knowing the costs of all links in the network. The distance vector algorithm we will now present is *decentralized* and does not use such global information. Indeed, the only information a node will have are the costs of the links to its directly attached neighbors, and information it receives from these directly attached neighbors. The distance vector algorithm we will study is also known as the Bellman-Ford algorithm, after its inventors. It is used in many routing algorithms in practice, including: Internet BGP, ISO IDRP, Novell IPX, and the original ARPAnet.

2-5-1 Distance Vector (DV) Algorithm. At each node, X:

```
1  Initialization:
2   for all adjacent nodes v:
3   $D^X(*,v)$ = infty        /* the * operator means "for all
rows" */
4      $D^X(v,v)$ = c(X,v)
5   for all destinations, y
6      send $\min_w D(y,w)$ to each neighbor  /* w over all X's
neighbors */
7
8  loop
9    wait (until I see a link cost change to neighbor V
10         or until I receive update from neighbor V)
11
12   if (c(X,V) changes by d)
13     /* change cost to all dest's via neighbor v by d */
14     /* note: d could be positive or negative */
15     for all destinations y:  $D^X(y,V)$ =  $D^X(y,V)$ + d
16
17   else if (update received from V wrt destination Y)
18     /* shortest path from V to some Y has changed  */
19     /* V has sent a new value for its  $\min_w D^V(Y,w)$ */
20     /* call this received new value is "newval"     */
21     for the single destination y: $D^X(Y,V)$ = c(X,V) + newval
22
23   if we have a new $\min_w D^X(Y,w)$for any destination Y
24      send new value of $\min_w D^X(Y,w)$ to all neighbors
25
26  forever
```

The key steps are lines 15 and 21, where a node updates its distance table entries in response to either a change of cost of an attached link or the receipt of an update message from a neighbor. The other key step is line 24, where a node sends an update to its neighbors if its minimum cost path to a destination has changed.

Figure 2-4 illustrates the operation of the DV algorithm for the simple three node network shown at the top of the figure. The operation of the algorithm is illustrated in a synchronous manner, where all nodes simultaneously receive messages from their neighbors, compute new distance table entries, and inform their neighbors of any changes in their new least path costs. After studying this example, you should convince yourself that the algorithm operates correctly in an asynchronous manner as well, with node computations and update generation/reception occurring at any times.

The circled distance table entries in Figure 2-4 show the current least path cost to a destination. An entry circled in red indicates that a new minimum cost has been computed (in either line 4 of the DV algorithm (initialization) or line 21). In such cases an update message will be sent (line 24 of the DV algorithm) to the node's neighbors as represented by the red arrows between columns in Figure 2-4.

The leftmost column in Figure2-4 shows the distance table entries for nodes X, Y, and Z after the initialization step.

Let us now consider how node X computes the distance table shown in the middle column of Figure 4.2-4 after receiving updates from nodes Y and Z. As a result of receiving the updates from Y and Z, X computes in line 21 of the DV algorithm:

$$
\begin{aligned}
D^X(Y,Z) &= c(X,Z) + \min_w D^Z(Y,w) \\
&= 7 + 1 \\
&= 8 \\
D^{X(}Z,Y) &= c(X,Y) + \min_w D^Y(Z,w) \\
&= 2 + 1 \\
&= 3
\end{aligned}
$$

It is important to note that the only reason that X knows about the terms $\min_w D^Z(Y,w)$ and $\min_w D^Y(Z,w)$ is because nodes Z and Y have sent those values to X (and are received by X in line 10 of the DV algorithm). As an exercise, verify the distance tables computed by Y and Z in the middle column of Figure 2-4.

The value $D^{X(}Z,Y) = 3$ means that X's minimum cost to Z has changed from 7 to 3. Hence, X sends updates to Y and Z informing them of this new least cost to Z. Note that X need not update Y and Z about its cost to Y since this has not changed. Note also that Y's re-computation of its distance table in the middle column of Figure 2-4 *does* result in new distance entries, but *does not* result in a change of Y's least cost path to nodes X and Z. Hence Y does *not* send updates to X and Z.

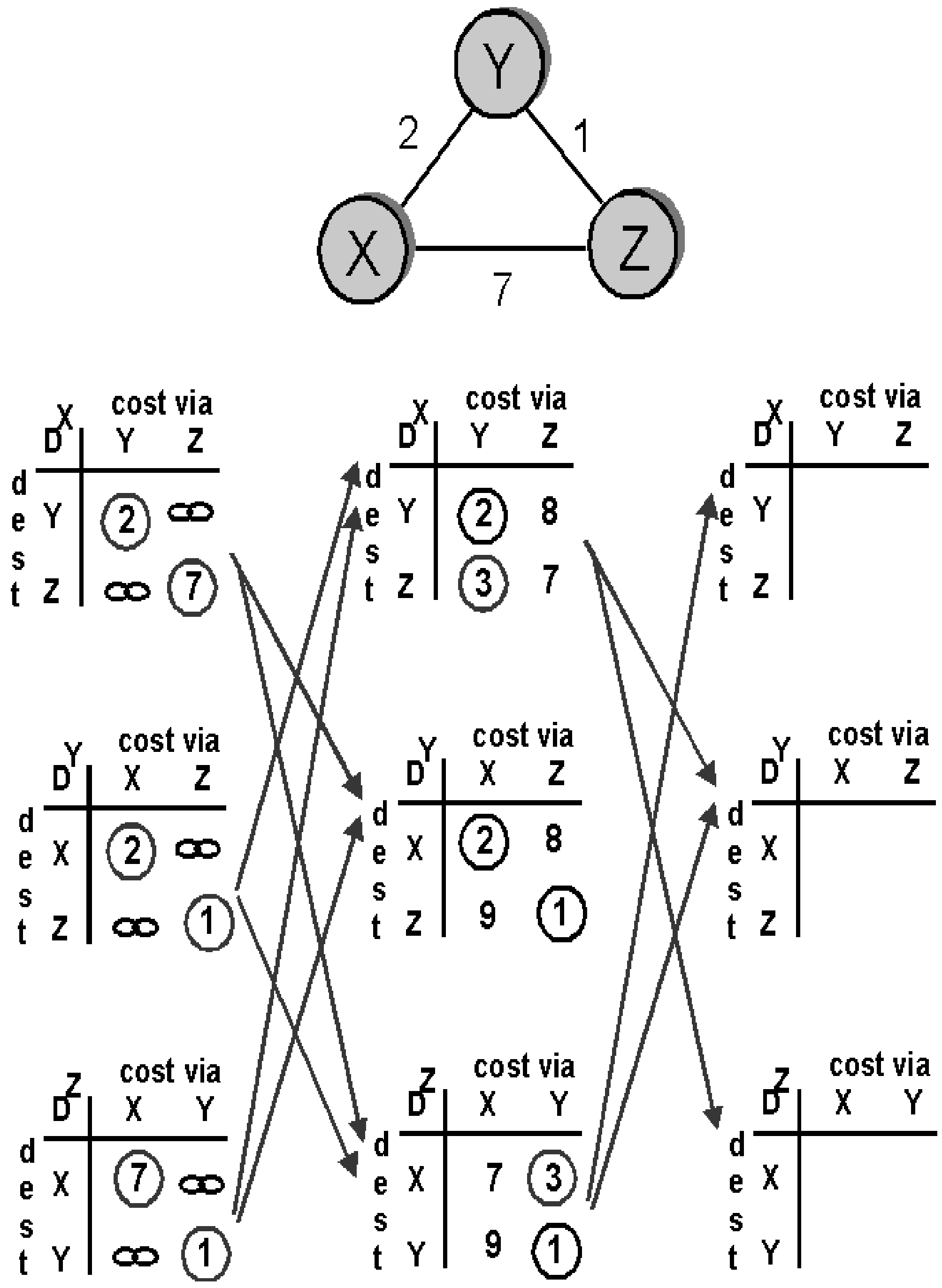

Y
X
Z
2
1
7
D^X cost via Y Z; dest Y: 2, ∞; Z: ∞, 7
D^Y cost via X Z; dest X: 2, ∞; Z: ∞, 1
D^Z cost via X Y; dest X: 7, ∞; Y: ∞, 1
D^X cost via Y Z; dest Y: 2, 8; Z: 3, 7
D^Y cost via X Z; dest X: 2, 8; Z: 9, 1
D^Z cost via X Y; dest X: 7, 3; Y: 9, 1
D^X cost via Y Z; dest Y; Z
D^Y cost via X Z; dest X; Z
D^Z cost via X Y; dest X; Y

Figure 2-4: Distance Vector Algorithm: example

The process of receiving updated costs from neighbors, re-computation of distance table entries, and updating neighbors of changed costs of the least cost path to a destination continues until no update messages are sent. At this point, since no update messages are sent, no further distance table calculations will occur and the algorithm enters a quiescent state, i.e., all nodes are performing the wait in line 9 of the DV algorithm. The algorithm would remain in the quiescent state until a link cost changes, as discussed below.

2-6 Distance Vector Algorithm: Link Cost Changes and Link Failure

When a node running the DV algorithm detects a change in the link cost from itself to a neighbor (line 12) it updates its distance table (line 15) and, if there is a change in the cost of the least cost path, updates its neighbors (lines 23 and 24). Figure 2-5 illustrates this behavior for a scenario where the link cost from Y to X changes from 4 to 1. We focus here only on Y and Z's distance table entries to destination (row) X.

- At time t_0, Y detects the link cost change (the cost has changed from 4 to 1) and informs its neighbors of this change since the cost of a minimum cost path has changed.
- At time t_1, Z receives the update from Y and then updates its table. Since it computes a new least cost to X (it has decreased from a cost of 5 to a cost of 2), it informs its neighbors.
- At time t_2, Y has receives Z's update and has updates its distance table. Y's least costs have not changed (although its cost to X via Z has changed) and hence Y does *not* send any message to Z. The algorithm comes to a quiescent state.

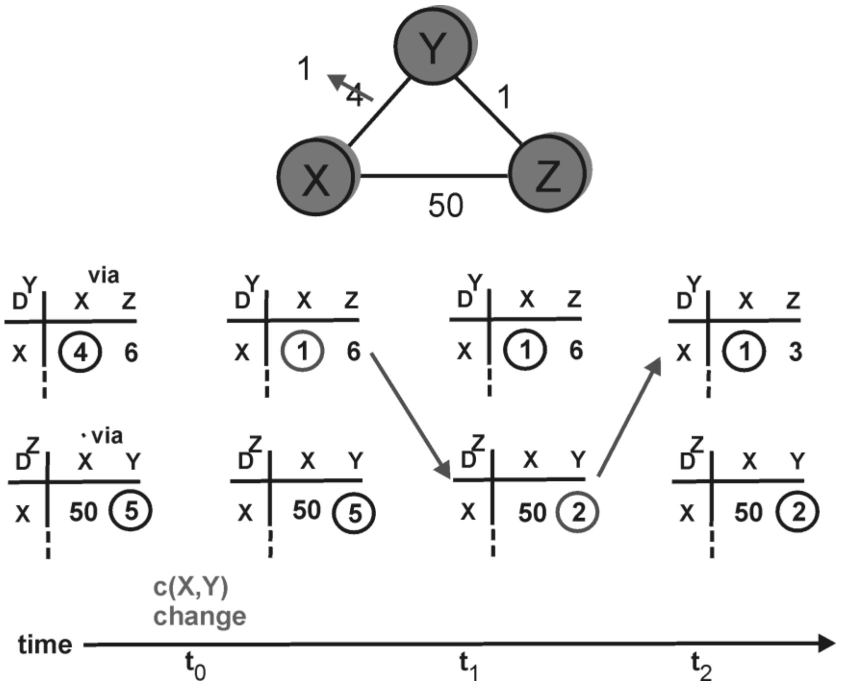


Figure 4.2-5: Link cost change: good news travels fast

In Figure 2-5, only two iterations are required for the DV algorithm to reach a quiescent state. The "good news" about the decreased cost between X and Y has propagated fast through the network.

Let's now consider what can happen when a link cost *increases.* Suppose that the link cost between X and Y increases from 4 to 60.

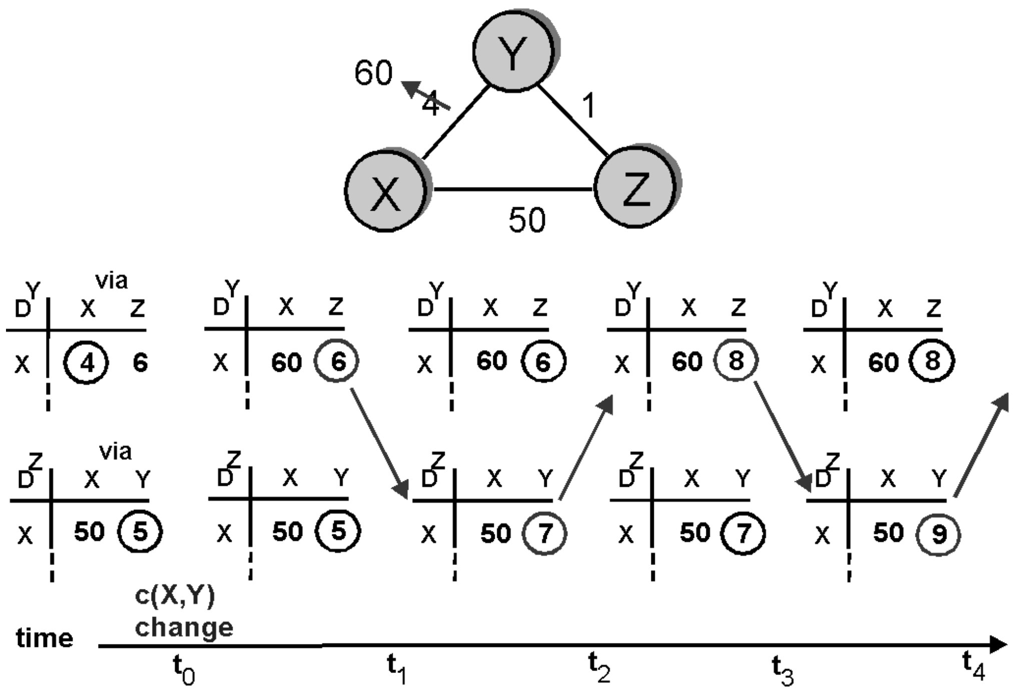


Figure 2-6: Link cost changes: bad news travels slow and causes loops

- At time t_0 Y detects the link cost change (the cost has changed from 4 to 60). Y computes its new minimum cost path to X to have a cost of 6 via node Z. Of course, with our global view of the network, we can see that this new cost via Z is *wrong.* But the only information node Y has is that its direct cost to X is 60 and that Z has last told Y that Z could get to X with a cost of 5. So in order to get to X, Y would now route through Z, fully expecting that Z will be able to get to X with a cost of 5. As of t_1 we have a **routing loop** in order to get to X, Y routes through Z, and Z routes through Y. A routing loop is like a black hole a packet arriving at Y or Z as of t_1 will bounce back and forth between these two nodes forever ..... Or until the routing tables are changed.
- Since node Y has computed a new minimum cost to X, it informs Z of this new cost at time t_1
- Sometime after t1, Z receives the new least cost to X via Y (Y has told Z that Y's new minimum cost is 6). Z knows it can get to Y with a cost of 1 and hence computes a new least cost to X (still via Y) of 7. Since

Y's least cost to X has increased, it then informs Y of its new cost at t_2.

- In a similar manner, Y then updates its table and informs Z of a new cost of 9. Z then updates its table and informs Y of a new cost of 10, etc...

How long will the process continue? You should convince yourself that the loop will persist for 44 iterations (message exchanges between Y and Z) until Z eventually computes its path via Y to be larger than 50. At this point, Z will (finally!) determine that its least cost path to X is via its direct connection to X. Y will then route to X via Z. The result of the "bad news" about the increase in link cost has indeed traveled slowly! What would have happened if the link cost change of c(Y,X) had been from 4 to 10,000 and the cost c(Z,X) had been 9,999? Because of such scenarios, the problem we have seen is sometimes referred to as the "count-to-infinity" problem.

2-6-1 Distance Vector Algorithm: Adding Poisoned Reverse.

The specific looping scenario illustrated in Figure 2-6 can be avoided using a technique known as poisoned reverse. The idea is simple if Z routes through Y to get to destination X, then Z will advertise to Y that its (Z's) distance to X is infinity. Z will continue telling this little "white lie" to Y as long as it routes to X via Y. Since Y believes that Z has no path to X, Y will never attempt to route to X via Z, as long as Z continues to route to X via Y (and lie about doing so).

Figure 2-7 illustrates how poisoned reverse solves the particular looping problem we encountered before in Figure 2-6. As a result of the poisoned reverse, Y's distance table indicates an infinite cost when routing to X via Z (the result of Z having informed Y that Z's cost to X was infinity). When the cost of the XY link changes from 4 to 60 at time t_0, Y updates its table and continues to route directly to X, albeit at a higher cost of 60, and informs Z of this change in cost. After receiving the update at t_1, Z immediately shifts it route to X to be via the direct ZX link at a cost of 50. Since this is a new least cost to X, and since the path no longer passes through Y, Z informs Y of this new least cost path to X at t_2. After receiving the update from Z, Y updates its distance table to route to X via Z at a least cost of 51. Also, since Z is now on Y's least path to X, Y poisons the reverse path from Z to X by informing Z at time t_3 that it (Y) has an infinite cost to get to X. The algorithm becomes quiescent after t_4, with distance table entries for destination X shown in the rightmost column in Figure 2-7.

Does poison reverse solve the general count-to-infinity problem? It does not. You should convince yourself that loops involving *three* or more nodes (rather than simply two immediately neighboring nodes, as we saw in Figure 2-7) will not be detected by the poison reverse technique.

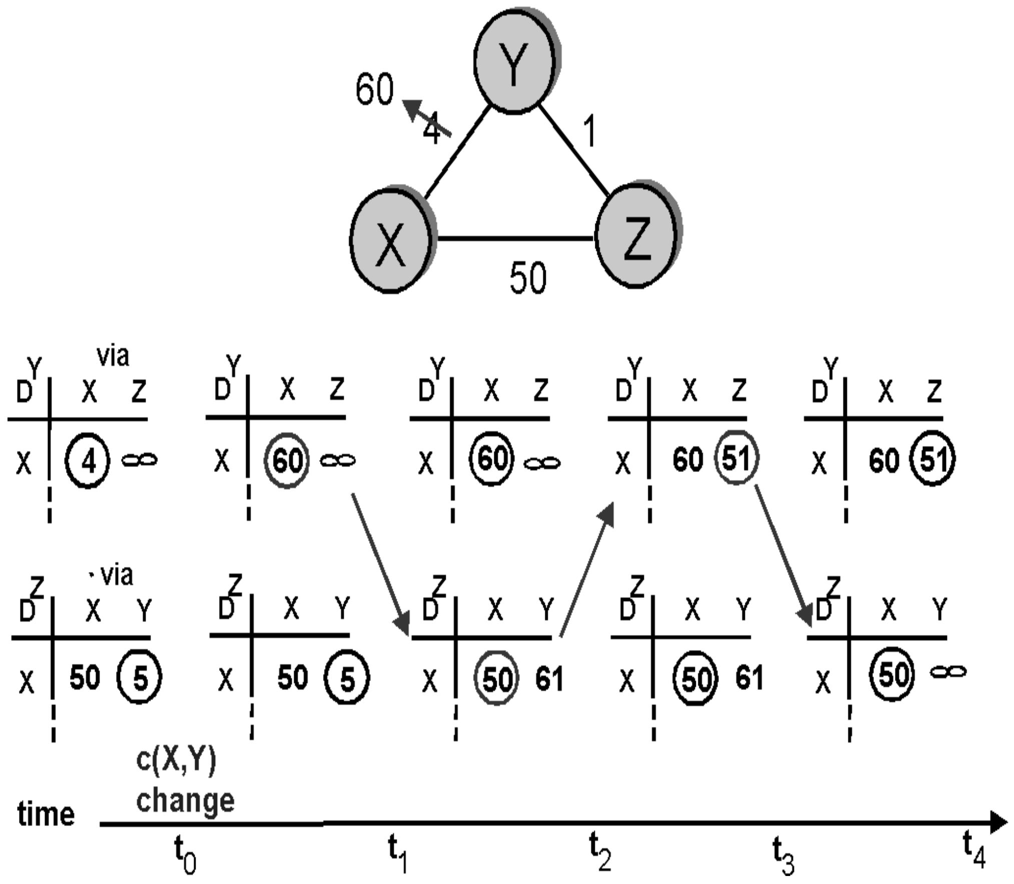


Figure 2-7: Poisoned reverse

2-7 A Comparison of Link State and Distance Vector Routing Algorithms

Let us conclude our study of link state and distance vector algorithms with a quick comparison of some of their attributes.

- **Message Complexity.** We have seen that LS requires each node to know the cost of each link in the network. This requires O(nE) messages to be sent, where n is the number of nodes in the network and E is the number of links. Also, whenever a link cost changes, the new link cost must be sent to *all* nodes. The DV algorithm requires message exchanges between directly connected neighbor at each iteration. We have seen that the time needed for the algorithm to converge can depend on many factors. When link costs change, the DV algorithm will propagate the results of the changed link cost *only* if the new link cost results in a changed least cost path for one of the nodes attached to that link.
- **Speed of Convergence.** We have seen that our implementation of the LS is an $O(n^2)$ algorithm requiring O(nE) messages, and potentially suffer from oscillations. The DV algorithm can converge

slowly (depending on the relative path costs, as we saw in Figure 2-7) and can have routing loops while the algorithm is converging. DV also suffers from the count to infinity problem.

- **Robustness.** What can happen is a router fails, misbehaves, or is sabotaged? Under LS, a router could broadcast an incorrect cost for one of its attached links (but no others). A node could also corrupt or drop any LS broadcast packets it receives as part of link state broadcast. But an LS node is only computing its own routing tables; other nodes are performing the similar calculations for themselves. This means route calculations are somewhat separated under LS, providing a degree of robustness. Under DV, a node can advertise incorrect least path costs to any/all destinations. (Indeed, in 1997 a malfunctioning router in a small ISP provided national backbone routers with erroneous routing tables. This caused other routers to flood the malfunctioning router with traffic, and caused large portions of the Internet to become disconnected for up to several hours.) More generally, we note that at each iteration, a node's calculation in DV is passed on to its neighbor and then indirectly to its neighbor's neighbor on the next iteration. In this sense, an incorrect node calculation can be diffused through the entire network under DV.

2-8 Other Routing Algorithms

The LS and DV algorithms we have studied are not only widely used in practice, they are essentially the only routing algorithms used in practice today.

Nonetheless, many routing algorithms have been proposed by researchers over the past 30 years, ranging from the extremely simple to the very sophisticated and complex. One of the simplest routing algorithms proposed is **hot potato routing.** The algorithm derives its name from its behavior -- a router tries to get rid of (forward) an outgoing packet as soon as it can. It does so by forwarding it on *any* outgoing link that is not congested, regardless of destination. Although initially proposed quite some time ago, interest in hot-potato-like routing has recently been revived for routing in highly structured networks, such as the so-called Manhattan street network.

Another broad class of routing algorithms is based on viewing packet traffic as flows between sources and destinations in a network. In this approach, the routing problem can be formulated mathematically as a constrained optimization problem known as a network flow problem. Let us define l_{ij} as the amount of traffic (e.g., in packets/sec) entering the network for the first time at node i and destined for node j. The set of flows, $\{l_{ij}\}$ for all i,j, is sometimes referred to as the network **traffic matrix.** In a network flow problem, traffic flows must be assigned to a set of network links subject to constraints such as:

- the sum of the flows between all source destination pairs passing though link m must be less than the capacity of link m;

- the amount of l_{ij} traffic entering any router r (either from other routers, or directly entering that router from an attached host) must equal the amount of l_{ij} traffic leaving router either via one of r's outgoing links or to an attached host at that router. This is a **flow conservatio**n constraint.

Let us define l_{ij}^m as the amount of source i, destination j traffic passing through link m. The optimization problem then is to find the set of link flows, $\{l_{ij}^m\}$ for all links m and all sources, i, and designations, j, that satisfies the constraints above and optimizes a performance measure that is a function of $\{l_{ij}^m\}$. The solution to this optimization problem then defines the routing used in the network. For example, if the solution to the optimization problem is such that $l_{ij}^m = l_{ij}$ for some link m, then all i-to-j traffic will be routed over link m. In particular, if link m is attached to node i, and then m is the first hop on the optimal path from source i to destination j.

But what performance function should be optimized? There are many possible choices. If we make certain assumptions about the size of packets and the manner in which packets arrive at the various routers, we can use the so-called M/M/1 queuing theory formula to express the average delay at link as:

$$D_m = 1 / (R_m - \Sigma_i \Sigma_j \lambda_{ij}^m),$$

where R_m is link m's capacity (measured in terms of the average number of packets/sec it can transmit) and $S_i S_j l_{ij}^m$ is the total arrival rate of packets (in packets/sec) that arrive to link m. The overall network wide performance measure to be optimized might then be the sum of all link delays in the network, or some other suitable performance metric. A number of elegant distributed algorithms exist for computing the optimum link flows (and hence routing determine the routing paths, as discussed above). The reader is referred to for a detailed study of these algorithms.

The final set of routing algorithms we mention here are those derived from the telephony world. These *circuit-switched* routing algorithms are of interest to packet-switched data networking in cases where per-link resources (e.g., buffers, or a fraction of the link bandwidth) are to reserved (i.e., set aside) for each connection that is routed over the link. While the formulation of the routing problem might appear quite different from the least cost routing formulation we have seen in this chapter, we will see that there are a number of similarities, at least as far as the path finding algorithm (routing algorithm) is concerned. Our goal here is to provide a brief introduction for this class of routing algorithms.

The circuit-switched routing problem formulation is illustrated in Figure 2-8. Each link has a certain amount of resources (e.g., bandwidth). The easiest (and a quite accurate) way to visualize this is to consider the link to be a bundle of circuits, with each call that is routed over the link requiring the dedicated use of one of the link's circuits. A link is thus characterized both by its total number of circuits, as well as the number of these circuits currently in use. In Figure 2-8, all links except AB and BD have 20 circuits; the number to the left of the number of circuits indicates the number of circuits currently in use.

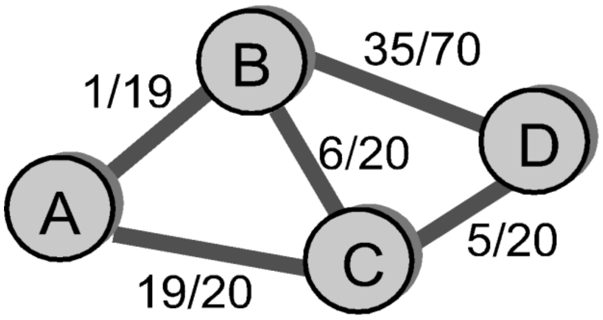


Figure 2-8: Circuit-switched routing

Suppose now that a call arrives at node A, destined to node D. What path should be taking? In **shortest path first** (SPF) routing, the shortest path (least number of links traversed) is taken. We have already seen how the Dijkstra LS algorithm can be used to find shortest path routes. In Figure 2-8, either that ABD or ACD path would thus be taken. In **least loaded path** (LLP) routing, the *load* at a link is defined as the ratio of the number of used circuits at the link and the total number of circuits at that link. The path load is the maximum of the loads of all links in the path. In LLP routing, the path taken is that with the smallest path load. In example 2-8, the LLP path is ABCD. In **maximum free circuit** (MFC) routing, the number of free circuits associated with a path is the minimum of the number of free circuits at each of the links on a path. In MFC routing, the path the maximum number of free circuits is taken. In Figure 2-8 the path ABD would be taken with MFC routing.

Given these examples from the circuit switching world, we see that the path selection algorithms have much the same flavor as LS routing. All nodes have complete information about the network's link states. Note however, that the potential consequences of old or inaccurate sate information are more severe with circuit-oriented routing a call may be routed along a path only to find that the circuits it had been expecting to be allocated are no longer available. In such a case, the call setup is blocked and another path must be attempted. Nonetheless, the main differences between connection-oriented, circuit-switched routing and connectionless packet-switched routing come not in the path selection mechanism, but rather in the actions that must be taken when a connection is set up, or torn down, from source to destination.

2-9 Hierarchical Routing

In the previous sections, we viewed "the network" simply as a collection of interconnected routers. One router was indistinguishable from another in the sense that all routers executed the same routing algorithm to compute routing paths through the entire network. In practice, this model and its view of a homogenous set of routers all executing the same routing algorithm is a bit simplistic for at least two important reasons:

Scale: As the number of routers becomes large, the overhead involved in computing, storing, and communicating the routing table information (e.g., link state updates or least cost path changes) becomes prohibitive. Today's public Internet consists of millions of interconnected routers and more than 50 million hosts. Storing routing table entries to each of these hosts and routers would clearly require enormous amounts of memory. The overhead required to broadcast link state updates among millions of routers would leave no bandwidth left for sending the data packets! A distance vector algorithm that iterated among millions of routers would surely never converge! Clearly, something must be done to reduce the complexity of route computation in networks as large as the public Internet.

Administrative autonomy: Although engineers tend to ignore issues such as a company's desire to run its routers as it pleases (e.g., to run whatever routing algorithm it chooses), or to "hide" aspects of the networks' internal organization from the outside, these are important considerations. Ideally, an organization should be able to run and administer its network as it wishes, while still being able to connect its network to other "outside" networks.

Both of these problems can be solved by aggregating routers into "regions" or "autonomous systems" (ASs). Routers within the same AS all run the same routing algorithm (e.g., a LS or DV algorithm) and have full information about each other exactly as was the case in our idealized model in the previous section. The routing algorithm running within an autonomous system is called an **intra-autonomous system routing protocol**. It will be necessary, of course, to connect ASs to each other, and thus one or more of the routers in an AS will have the added task for being responsible for routing packets to destinations outside the AS. Routers in an AS that have the responsibility of routing packets to destinations outside the AS are called **gateway routers**. In order for gateway routers to route packets from one AS to another (possibly passing through multiple other ASs before reaching the destination AS), the gateways must know how to route (i.e., determine routing paths) among themselves. The routing algorithm that gateways use to route among the various ASs is known as an **inter-autonomous system routing protocol.**

In summary, the problems of scale and administrative authority are solved by defining autonomous systems. Within an AS, all routers run the same intra-autonomous system routing protocol. Special gateway routers in the various ASs run an inter-autonomous system routing protocol that determines routing paths among the ASs. The problem of scale is solved since an intra-AS router need only know about routers within its AS and the gateway router(s) in it’s AS. The problem of administrative authority is solved since an organization can run whatever intra-AS routing protocol it chooses, as long as the AS's gateway(s) is able to run an inter-AS routing protocol that can connect the As to other ASs..

Figure 2-9 illustrates this scenario. Here, there are three routing ASs, A, B and C. Autonomous system A has four routers, A.a, A.b, A.c and A.d, which run the intra-AS routing protocol used within autonomous system A. These four routers have complete information about routing paths within autonomous system A. Similarly, autonomous systems B and C have three and two routers, respectively. Note that the intra-AS routing protocols running in A, B and C need not be the same. The gateway routers are A.a, A.c, B.a and C.b. In addition to running the intra-AS routing protocol in conjunction with other routers in their ASs, these four routers run an inter-AS routing protocol among themselves. The topological view they use for inter-AS routing protocol is shown at the higher level, with "links" shown in light gray. Note that a "link" at the higher layer may be an actual physical link, e.g., the link connection A.c and B.a, or a logical link, such as the link connecting A.c and A.a.

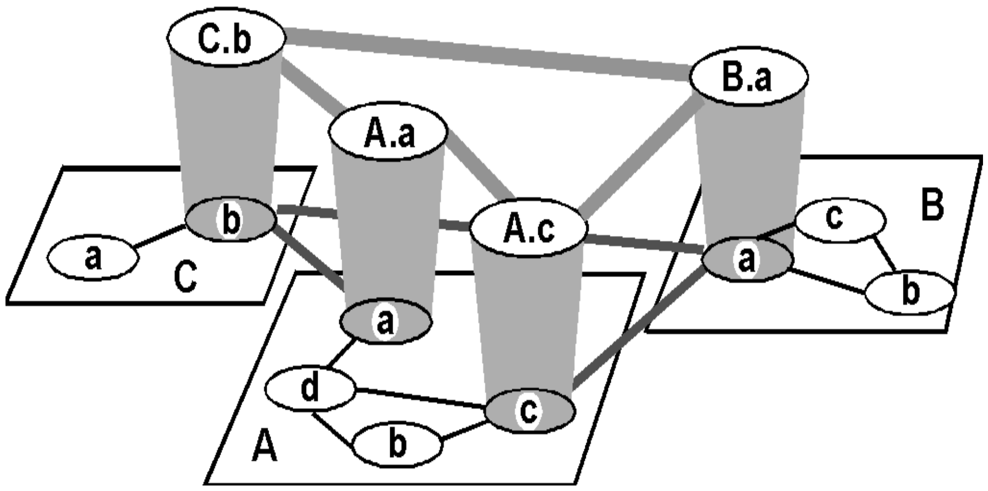


Figure 2-9: Intra-AS and Inter-AS routing.

Figure 2-10 illustrates that the gateway router A.c must run an intra-AS routing protocol with its neighbors A.b and A.d, as well as an inter-AS protocol with gateway router B.a.

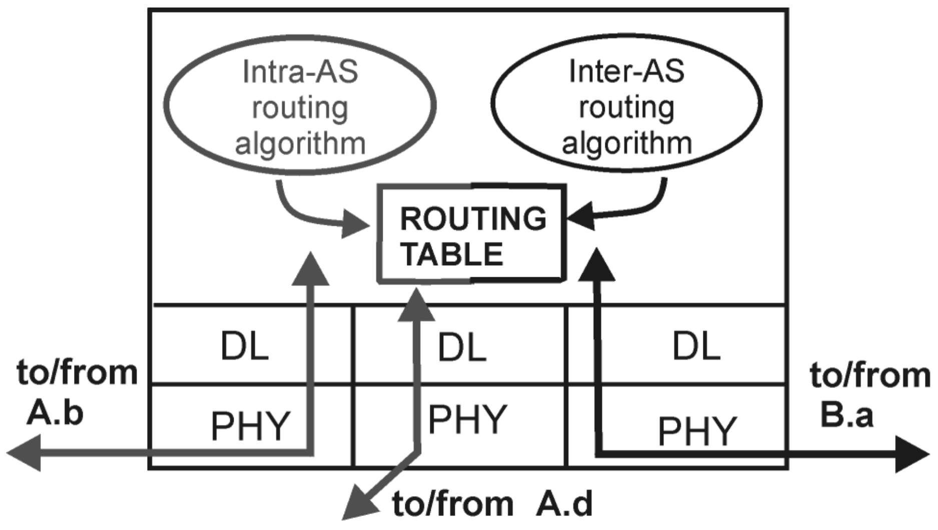


Figure 2-10: Internal architecture of gateway router A.c

Suppose now that a host h1 attached to router A.d needs to route a packet to destination h2 in autonomous system B, as shown in Figure 2-11. Assuming that A.d's routing table indicates that router A.c is responsible for routing its (A.d's) packets outside the AS, the packet is first routed from A.d to A.c using A's intra-AS routing protocol. It is important to note that router A.d does not know about the internal structure of autonomous systems B and C and indeed need not even know about the topology connecting autonomous systems A, B and C. Router A.c will receive the packet and see that it is destined to an autonomous system outside of A. A's routing table for the intra-AS protocol would indicate that a packet destined to autonomous system B should be routed along the A.c to B.a link. When the packet arrives at B.a, B.a's *inter-AS* routing sees that the packet is destined for autonomous system B. The packet is then "handed over" to the *intra-AS* routing protocol within B, which routes the packet to its final destination, h2.

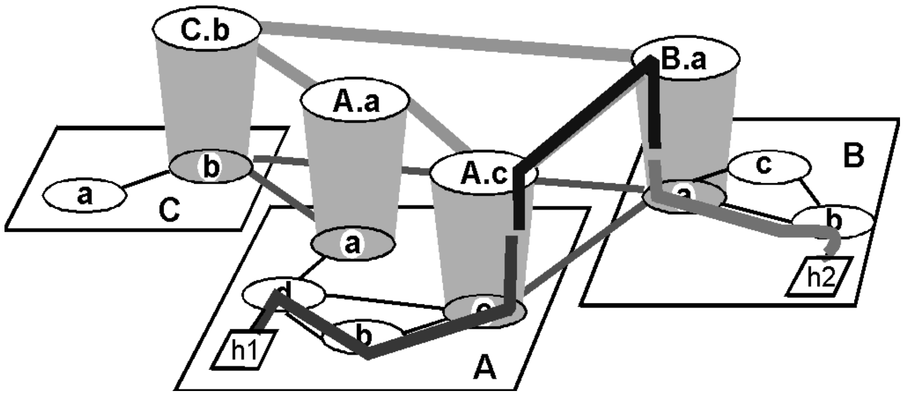


Figure 2-11: The route from A.d to B.b: intra-AS and inter-AS path segments.

3. Internet Protocol

Structure

Objectives

- Discuss the field format of IP datagram, IP addressing;
- Describe IP Fragmentation and Reassembly
- Introduce Internet Control Message Protocol (ICMP).

3-1 Introduction

So far in this unit we have examined the underlying principles of the network layer. We have discussed the network layer service models, including virtual circuit service and datagram service, the routing algorithms commonly used to determine paths between origin and destination hosts, and how problems of scale are addressed with hierarchical routing. We are now going to turn our attention to the Internet's network layer.

As we mentioned in lesson-1, the Internet's network layer does not provide a virtual-circuit service, but instead a connectionless datagram service. When the network layer at the sending host receives a segment from the transport layer, it encapsulates the segment within an IP datagram, writes the destination address of the host (as well as other fields) on the datagram, and drops the datagram into the network. This process is similar to a person writing a letter, inserting the letter in an envelope, writing the destination address on the envelope, and dropping the envelope into a mailbox. Neither the Internet's network layer nor the postal service make any kind of preliminary contact with the destination before moving its "parcel" to the destination. Furthermore, as discussed in lesson-1, the network layer service is a best effort service. It does not guarantee that the datagram will arrive within a certain time, it does not guarantee that a series of datagram’s will arrive in the same order sent; in fact, it does not even guarantee that the datagram will ever arrive at its destination.

As we discussed in lesson-1, the network layer for a datagram network, such as the Internet, has two major components. First, it has a network protocol component, which defines network-layer addressing, the fields in the datagram (i.e., the network layer PDU), and how the end systems and routers act on these fields. The network protocol in the Internet is called the Internet Protocol, or more commonly, the **IP Protocol**. There are currently two versions of the IP protocol in use today. In this section we examine the more widespread version, namely, Internet Protocol version 4, which is specified in and which is more commonly known as IPv4. In next lesson we shall examine, IPv6, which is expected to slowly replace IPv4 in the upcoming years. The second major component of the network layer is the path determination component, which determines the route a datagram follows from origin to destination. We study the path determination component in the next section.

3-2 IP Addressing

Before discussing IP addressing, we need to say a few words about hosts and routers. A host (also called an end system) has one link into the network. When IP in the host wants to send a datagram, it passes the datagram to its link. The boundary between the host and the link is called the **interface**. A router is fundamentally different from a host in that it has two or more links that connect to it. When a router forwards a datagram, it forwards the datagram over one of its links. The boundary between the router and any one of its links is also called an **interface**. Thus, a router has multiple interfaces, one for each of its links. Because every interface (for a host or router) is capable of sending and receiving IP datagram's, IP requires each interface to have an IP address.

Each IP address is 32 bits long (equivalently, four bytes) long. IP addresses are typically written in so-called "dot-decimal notation", whereby each byte of the address is written in its decimal form and is separated by a period. For example, a typical IP address would be 193.32.216.9. The 193 is the decimal equivalent for the first 8 bits of the address; the 32 is the decimal equivalent for the second 8 bits of the address, etc. Thus, the address 193.32.216.9 in binary notation is:

11000001 00100000 11011000 00001001

A space has been added between the bytes for visual purposes. Because each IP address is 32 bits long, there are 2^{32} possible IP addresses.

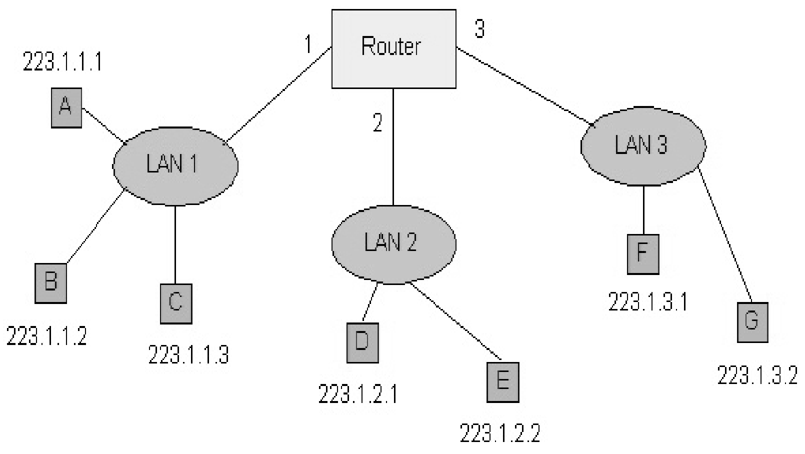


Figure 3-1: LANs are networks in IP jargon.

Figure 3-1 provides an example of IP addressing and interfaces. In this figure there is one router which interconnects three LANs. (LANs, also known as local area networks, were briefly discussed in unit 1 and will be studied in detail in the next chapter.) In the jargon of IP, each of these LANs is called an **IP network** or more simply a **"network"**. There are several things to observe from this diagram. First, the router has threes interfaces, labeled 1, 2 and 3. Each of the router interfaces has its own IP address, which is provided in Figure 4.4-2; each host also has its own interface and IP address. Second, all of the interfaces attached to LAN 1, including a router interface, have an IP address of the form 223.1.1.xxx. Similarly, all the interfaces attached to LAN 2 and LAN 3 has IP addresses of the form 223.1.2.xxx and 233.1.3.xxx, respectively. In other words, each address has two parts: the first part (the first three bytes in this example) that specifies the network; and the second part (the last byte in this example) that addresses a specific host on the network.

Router Interface	IP Address
1	223.1.1.4
2	223.1.2.9
3	223.1.3.27

Figure 3-2: IP addresses for router interfaces.

The IP definition of a "network" is not restricted to a LAN. To get some insight here, let us now take a look at another example. Figure 3-3 shows several LANs interconnected with three routers. All of the interfaces attached to LAN 1, including the router R1 interface that is attached to LAN 1, have an IP address of the form 223.1.1.xxx. Similarly, all the interfaces attached to LAN 2 and to LAN 3 have the form 223.1.2.xxx and 223.1.3.xxx, respectively. Each of the three LANs again constitutes their own network (i.e., IP network). But note that there are three additional "networks" in this example: one network for the interfaces that connect Router 1 to Router 2; another network for the interfaces that connect Router 2 to Router 3; and a third network for the interfaces that connect Router 3 to Router 1.

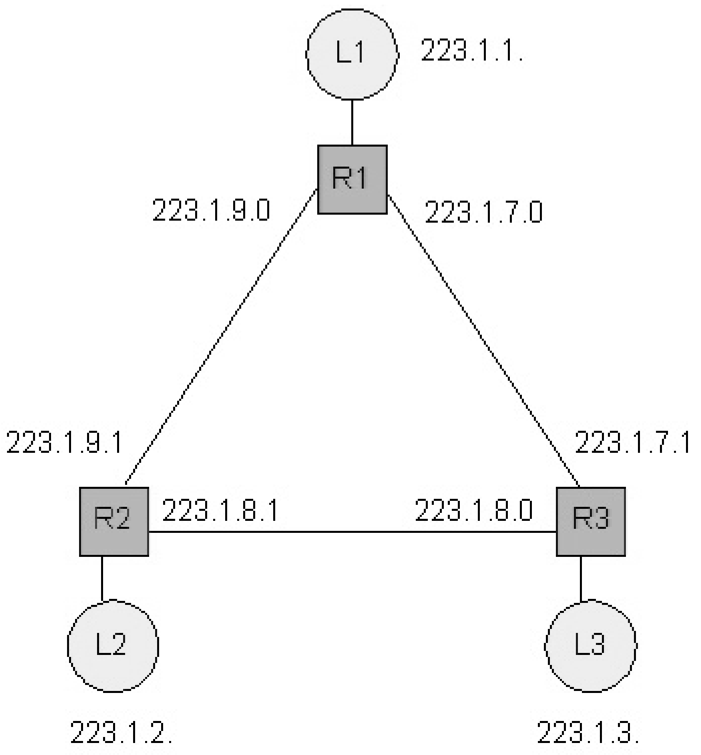


Figure 3-3: An interconnected system consisting of six networks.

For a general interconnected system of routers and hosts (such as the Internet), we use the following recipe to define the "networks" in the system. We first detach each router interface from its router and each host interface from its host. This creates "islands" of isolated networks, with "interfaces" terminating all the leaves of the isolated networks. We then call each of these isolated networks a **network**. Indeed, if we apply this procedure to the interconnected system in Figure 3-3, we get six islands or "networks". The current Internet consists of millions of networks. (In the next chapter we will consider bridges. We mention here that when applying this recipe, we do not detach interfaces from bridges. Thus each bridge lies within the interior of some network.)

Now that we have defined a network, we are ready to discuss IP addressing in more detail. IP addresses are globally unique, that is, no two interfaces in the world have the same IP address. Figure 3-3 shows the four possible formats of an IP address. (A fifth address, beginning with 11110, is reserved for future use.) In general, each interface (for a host or router) belongs to a network; the network part of the address identifies the network to which the interface belongs. The host part identifies the specific interface within the network. (We would prefer to use the terminology "interface part of the address" rather than "host part of the address" because IP address is really for an interface and not for a host; but the terminology "host part" is commonly used in practice.) For a class A address, the first 8 bits identify the network, and the last 24 bits identify the interface within that network. Thus with a class A we can have up to 2^7 networks (the first of the eight bits is fixed as 0) and 2^{24} interfaces. Note that the interfaces in Figures 3.1 and 3.3 use class A addresses. The class B address space allows for 2^{14} networks, with up to 2^{16} interfaces within each network. A class C address uses 21 bits to identify the network and leaves only 8 bits for the interface identifier. Class D addresses are reserved for so-called multicast addresses. As we will see in lesson 5, these addresses do not identify a specific interface but rather provide a mechanism through which multiple hosts can receive a copy of each single packet sent by a sender.

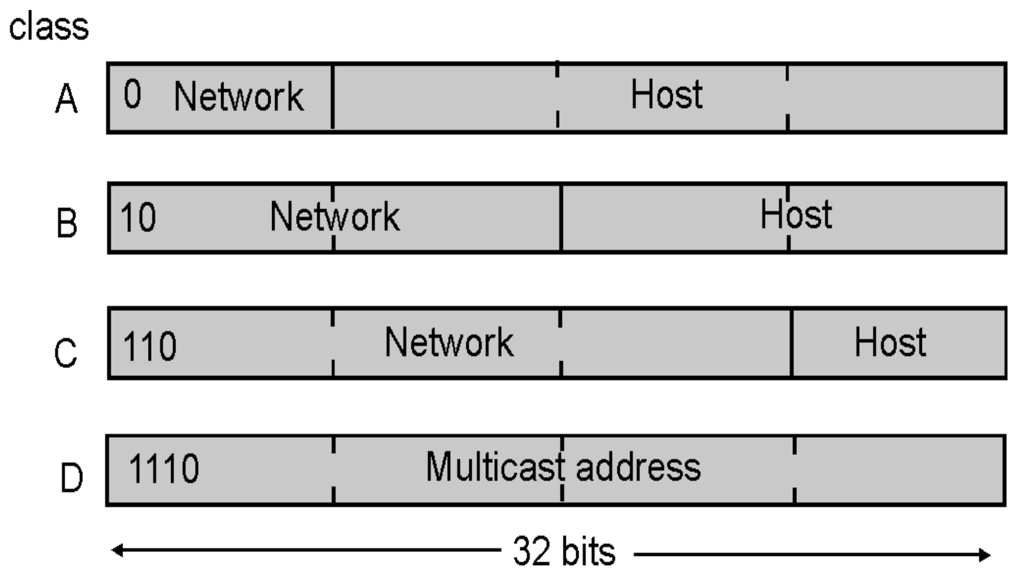


Figure 3-4: IPv4 address formats.

3-2-1 Assigning Addresses

Having introduced IP addressing, one question that immediately comes to mind is how does a host get its own IP address? We have just learned that an IP address has two parts, a network part and a host part. The host part of the address can be assigned in several different ways, including:

- **Manual configuration:** The IP address is configured into the host (typically in a file) by the system administrator.

- **Dynamic Host Configuration Protocol (DHCP):** DHCP is an extension of the BOOTP protocol, and is sometimes referred to as Plug and Play. With DHCP, a DHCP server in a network (e.g., in a LAN) receives DHCP requests from a client and in the case of dynamic address allocation, allocates an IP address back to the requesting client. DHCP is used extensively in LANs and in residential Internet access.

The network part of the address is the same for all the hosts in the network. To obtain the network part of the address for a network, the network administrator might first contact the network's ISP, which would provide addresses from a larger block of addressees that have already been allocated to the ISP. But how does an ISP get a block of addresses? IP addresses are managed under the authority of the Internet Assigned Numbers Authority (IANA), under the guidelines set forth in [RFC 2050]. The actual assignment of addresses is now managed by regional Internet registries. As of mid-1998, there are three such regional registries: the American Registry for Internet Number (ARIN, which handles registrations for North and South America, as well as parts of Africa. ARIN has recently taken over a number of the functions previously provided by Network Solutions), the Reseaux IP Europeans (RIPE, which covers Europe and nearby countries), and the Asia Pacific Network Information Center (APNIC).

Before leaving our discussion of addressing, we want to mention that mobile hosts may change the network to which they are attached, either dynamically while in motion or on a longer time scale. Because routing is to a network first, and then to a host within the network, this means that the mobile host's IP address must change when the host changes networks. Techniques for handling such issues are now under development within the IETF and the research community [RFC2002] [RFC2131].

3-3 The Big Picture: Transporting a Datagram from Source to Destination

Now that we have defined interfaces and networks, and that we have a basic understanding of IP addressing, we take a step back and discuss how IP transports a datagram from source to destination. To this end, a high level view of an IP datagram is shown in Figure 3-5. Note that every IP datagram has a destination address field and a source address field. The source host fills the source address field with its own 32-bit IP address and fills the destination address field with the 32-bit IP address of the host to which it wants to send the datagram. Note that these actions are analogous to what you do when you send a letter: on the envelope of the letter, you provide a destination address and a return (source) address. The data field of the datagram is typically filled with a TCP or UDP segment. We will discuss the remaining IP datagram fields a little later in this section.

NOTES

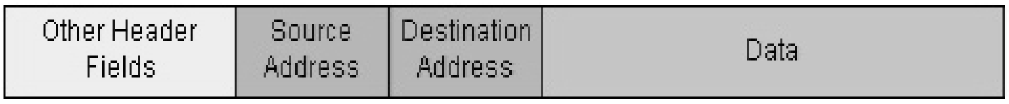


Figure 3-5: The key fields in an IP datagram.

Once the source host creates the IP datagram, how does the network layer transport the datagram from the source host to the destination host? Let us answer this question in the context of network Figure 3-1. First suppose host A wants to send an IP datagram to host B. The datagram is transported from host A to host B as follows. IP in host A first extracts the network portion of the address, 223.1.1. , and scans its *routing table*, which is shown in Figure 4.4-6. In this table, the "number of hops to destination" is defined to be the number of networks that need to be traversed, including the destination network. Scanning the table, host A finds a match in the first row, and observes that the number of hops to the destination is 1. This indicates to host A that the destination host is on the same network. Host A then passes the IP datagram to the link layer protocol and indicates to the link layer protocol that the destination is on the same LAN. The link layer protocol then has the responsibility of transporting the datagram to host B. (We will study how the link layer transports a datagram between to interfaces on the same network in the next chapter.)

destination network	**next router**	**number of hops to destination**
223.1.1.	-	1
223.1.2.	223.1.1.4	2
223.1.3.	223.1.1.4	2

Figure 3-6: Routing table in host A.

Now consider the more interesting case of host A sending an IP datagram to host E, which has IP address 223.1.2.2 and is on a different LAN. Host A again scans its routing table, but now finds a match in the second row. Because the number of hops to the destination is 2, host A knows that the destination is on another network. The routing table also tells host A that in order to get the datagram to host E, host A should first send the datagram to router address 223.1.1.4. IP in host A then passes the datagram down to the link layer, and indicates to the link layer that it should first send the datagram to IP address 223.1.1.4 .The link layer then transports the datagram to the router interface 1. The datagram is now in the router, and it is the job the router to move the datagram towards the datagram's ultimate destination. The router extracts the network portion of the destination address of the IP datagram, namely 223.1.2. , and scans its routing table, which is shown in Figure 3-7. The router finds a match in the second row of the table. The table tells the router that the datagram should be forwarded on router interface 2; also the number of hops to the destination is 1, which indicates to the router that the destination host is on the LAN directly attached to interface 2. The router moves the datagram to interface 2. Once the datagram is at interface 2, the router passes the datagram to link layer protocol and indicates to the link layer protocol that the destination host is on the same LAN. The link layer protocol has the job of transporting the datagram from the router interface 2 to host E, both of which are attached to the same LAN.

destination network	next router	number of hops to destination	interface
223.1.1.	-	1	1
223.1.2.	-	1	2
223.1.3.	-	1	3

Figure 3-7: Routing table in router.

In Figure 3-7, note that the entries in the "next router" column are all empty. This is because all of the networks (223.1.1., 223.1.2., and 223.1.3.) are each directly attached to the router, that is, there is no need to go through an intermediate router to get to the destination host. However, if host A and host E were separated by two routers, then within the routing table of the first router along the path from A to B, the appropriate row would indicate 2 hops to the destination and would specify the IP address of the second router along the path. The first router would then forward the datagram to the second router, using the link layer protocol that connects the two routers. The second router then forwards the datagram to the destination host, using the link layer protocol that connects the second router to the destination host.

You may recall from unit 1 that we said that routing a datagram in the Internet is similar to a person driving a car and asking gas station attendants at each intersection along the way how to get to the ultimate destination. It should now be clear why this appropriate analogy for routing in the Internet. As a datagram travels from source to destination, it visits a series of routers. At each router in the series, it stops and asks the router how to get to its ultimate destination. Unless the router is on the same LAN as the ultimate destination, the routing table essentially says to the datagram: "I don't know exactly how to get to the ultimate destination, put I do know that the ultimate destination is in the direction of the link (analogous to a road) connected to interface 3." The datagram then sets out on the link connected to interface 3, arrives at a new router, and again asks for new directions.

From this discussion we see that the routing tables in the routers play a central role in routing datagram's through the Internet. But how are these routing tables configured and maintained for large networks with multiple paths between sources and destinations (such as in the Internet)? Clearly, these routing tables should be configured so that the datagram's follow good (if not optimal) routes from source to destination. As you probably guessed, routing algorithms - like those studied in lesson 2 have the job of configuring and maintaining the routing tables. Furthermore, as discussed in lesson 2(session2-9), the Internet is partitioned into autonomous systems (ASs): intra-AS routing algorithms independently configure the routing tables within the autonomous systems; inter-AS routing algorithms have the job configuring routing tables so that datagram's can pass through multiple autonomous systems. We will discuss the Internet's intra-AS and inter-AS routing algorithms in lesson 4. But before moving on to routing algorithms, we cover three more important topics for the IP protocol, namely, the datagram format, datagram fragmentation, and the Internet Control Message Protocol (ICMP).

3-4 Datagram Format

The IPv4 datagram format is shown in Figure 3-8.

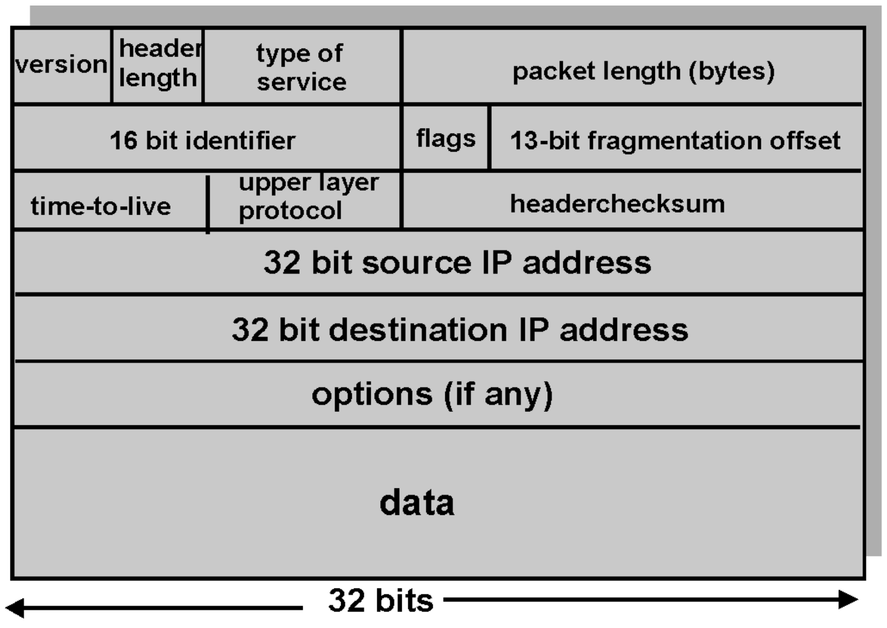


Figure 3-8: IPv4 datagram format

The key fields in the IPv4 datagram are the following:

- **Version Number:** These 4 bits specify the IP protocol version of the datagram. By looking at the version number, the router can then determine how to interpret the remainder of the IP datagram. Different versions of IP use different datagram formats. The datagram format for the "current" version of IP, IPv4, is shown in Figure 3-8. The datagram format for the "new" version of IP (IPv6).
- **Header Length:** Because an IPv4 datagram can contain a variable number of options (which are included in the IPv4 datagram header) these 4 bits are needed to determine where in the IP datagram the data actually begins. Most IP datagram's do not contain options so the typical IP datagram has a 20 byte header.
- **TOS:** The type of service (TOS) bits were included in the IPv4 header to allow different "types" of IP datagram's to be distinguished from each other, presumably so that they could be handled differently in times of overload. When the network is overloaded, for example, it would be useful to be able to distinguish network control datagram's from datagram's carrying data (e.g., HTTP messages). It would also be useful to distinguish real-time datagram's (e.g., used by an IP telephony application) from non-real-time traffic (e.g., FTP). More recently, one major routing vendor (Cisco) interprets the first three ToS bits as defining differential levels of service that can be provided by the router. The specific level of service to be provided is a policy issue determined by the router's administrator. **Datagram Length:** This is the total length of the IP datagram (header plus data) measured in bytes. Since this field is 16 bits long, the theoretical maximum size of the IP datagram to 65,535 bytes. However, datagram's are rarely greater than 1500 bytes, and are often limited in size to 576 bytes.
- **Identifier, Flags, Fragmentation Offset:** These three fields have to do with so-called IP fragmentation, a topic we will consider in depth shortly. Interestingly, the new version of IP, IPv6, simply does not allow for fragmentation.
- **Time-to-live:** The time-to-live (TTL) field is included to insure that datagram do not circulate forever (due to, for example, a long lived router loop) in the network. This field is decremented by one each time the datagram is processed by a router. If the TTL field reaches 0, the datagram must be dropped.
- **Protocol:** This field is only used when an IP datagram reaches its final destination. The value of this field indicates the transport-layer protocol at the destination to which the data portion of this IP datagram will be passed. For example, a value of 6 indicates that the data portion is passed to TCP, while a value of 17 indicates that the data is passed to UDP. For a listing of all possible numbers, see [RFC 1700]. Note that the protocol number in the IP datagram has a role that is fully analogous to the role of the port number field in the

transport-layer segment. The protocol number is the "glue" that holds the network and transport layers together, whereas port number is the "glue" that holds the transport and application layers together. We will see in Chapter 5 that the link layer frame also has a special field which glues the link layer to the network layer.

- **Header Checksum:** The header checksum aids a router in detecting bit errors in a received IP datagram. The header checksum is computed by treating each 2 bytes in the header as a number and summing these numbers using 1's complement arithmetic. As discussed in Section 3.3, the 1's complement of this sum, known as the Internet checksum, is stored in the checksum field. A router computes the Internet checksum for each received IP datagram and detects an error condition if the checksum carried in the datagram does not equal the computed checksum. Routers typically discard datagram's for which an error has been detected. Note that the checksum must be recomputed and restored at each router, as the TTL field, and possibly options fields as well, may change. An interesting discussion of fast algorithms for computing the Internet checksum is. A question often asked at this point is, why does TCP/IP perform error checking at both the transport and network layers? There are many reasons for this. First, routers are not required to perform error checking, so the transport layer cannot count on the network layer to do the job. Second, TCP/UDP and IP do not necessarily have to both belong to the same protocol stack. TCP can, in principle, run over a different protocol (e.g., ATM) and IP can carry data without passing through TCP/UDP (e.g., RIP data).
- **Source and Destination IP Address:** These fields carry the 32 bit IP address of the source and final destination for this IP datagram. The use and importance of the destination address is clear. The source IP address (along with the source and destination port numbers) is used at the destination host to direct the application data in the proper socket.
- **Options:** The optional options fields allow an IP header to be extended. Header options were meant to be used rarely hence the decision to save overhead by not including the information in options fields in every datagram header. However, the mere existence of options does complicate matters since datagram headers can be of variable length, one can not determine a priori where the data field will start. Also, since some datagram's may require options processing and others may not, the amount of time needed to process an IP datagram can vary greatly. These considerations become particularly important for IP processing in high performance routers and hosts. For these reasons and others, IP options were dropped in the IPv6 header.
- **Data (payload):** Finally, we come to the last, and most important field - the *raison d'être* for the datagram in the first place! In most circumstances, the data field of the IP datagram contains the transport-layer segment (TCP or UDP) to be delivered to the

destination. However, the data field can carry other types of data, such ICMP messages Note that IP datagram has a total of 20 bytes of header (assuming it has no options). If the IP datagram carries a TCP segment, then each (non-fragmented) datagram carries a total of 40 bytes of header (20 IP bytes and 20 TCP bytes) along with the application-layer data.

3-5 IP Fragmentation and Reassembly

We will see in unit 4 that not all link layer protocols can carry packets of the same size. Some protocols can carry "big" packets whereas other protocols can only carry "little" packets. For example, Ethernet packets can carry no more than 1500 bytes of data, whereas packets for many wide-area links can carry no more than 576 bytes. The maximum amount of data that a link-layer packet can carry is called the **MTU (maximum transfer unit)**. Because each IP datagram is encapsulated within the link-layer packet for transport from one router to the next router, the MTU of the link-layer protocol places a hard limit on the length of an IP datagram. Having a hard limit on the size of an IP datagram is not much of a problem. What is a problem is that each of the links along the route between sender and destination can use different link-layer protocols, and each of these protocols can have different MTUs.

To understand the problem better, imagine that *you* are a router that interconnects several links, each running different link-layer protocols with different MTUs. Suppose you receive an IP datagram from one link, you check your routing table to determine the outgoing link, and this outgoing link has an MTU that is smaller than the length of the IP datagram. Time to panic how are you going to squeeze this oversized IP packet into the payload field of the link-layer packet? The solution to this problem is to "fragment" the data in the IP datagram among two or smaller IP datagram’s, and then send these smaller datagram’s over the outgoing link. Each of these smaller datagram’s is referred to as a **fragment**.

Fragments need to be reassembled before they reach the transport layer at the destination. Indeed, both TCP and UDP are expecting to receive from the network layer complete, un-fragmented segments. The designers of IPv4 felt that reassembling (and possibly re-fragmenting) datagram’s in the routers would introduce significant complication into the protocol and put a damper on router performance. (If you were a router, would you want to be reassembling fragments on top of everything else you have to do?) Sticking to end-to-end principle for the Internet, the designers of IPv4 decided to put the job of datagram reassembly in the end systems rather than in the network interior.

When a destination host receives a series of datagram’s from the same source, it needs to determine if any of these datagram’s is fragments of some "original" bigger datagram. If it does determine that some datagram’s are fragments, it must further determine when it has received the last fragment and how the fragments it has received should be pieced back together to form the original datagram. To allow the destination host to perform these reassembly tasks, the designers of IP (version 4) put identification*, flag* and *fragmentation* fields in the IP datagram. When a datagram is created, the sending host stamps the datagram with an identification number as well as a source and destination address. The sending host increments the identification number for each datagram it sends. When a router needs to fragment a datagram, each resulting datagram (i.e., "fragment") is stamped with the source address, destination address and identification number of the original datagram. When the destination receives a series of datagram's from the same sending host, it can examine the identification numbers of the datagram's to determine which of the datagram's are actually fragments of the same bigger datagram. Because IP is an unreliable service, one or more of the fragments may never arrive at the destination. For this reason, in order for the destination host to be absolutely sure it has received the last fragment of the original datagram, the last fragment has a flag bit set to 0 whereas all the other fragments have this flag bit set to 1. Also, in order for the destination host to determine if a fragment is missing (and also to be able to reassemble the fragments in the proper order), the offset field is used to specify where the fragment fits within the original IP datagram. This bit is set to 1 in all except the last fragment.

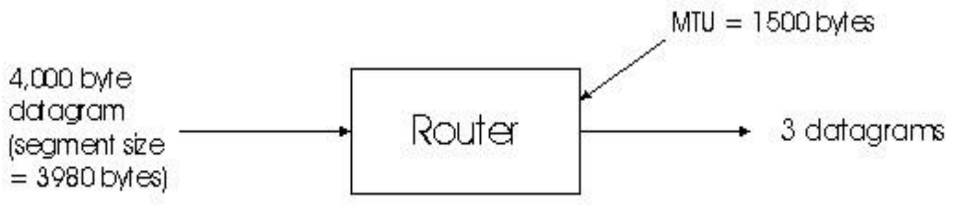


Figure 3-9: IP Fragmentation

Figure 3-9 illustrates an example. A datagram 4,000 bytes arrives to a router and this datagram must be forwarded to a link with a MTU of 1500 bytes. These imply that the 3,980 data bytes in the original datagram must be allocated to three separate fragments (each of which are also IP datagram's). Suppose that the original datagram is stamped with an identification number of 777. Then the characteristics of the three fragments are as follows:

1st fragment

- 1480 bytes in the data field of the IP datagram.
- identification = 777
- offset = 0 (meaning the data should be inserted beginning at byte 0)
- flag = 1 (meaning there is more)

2nd fragment

- 1480 byte information field
- identification = 777
- offset = 1,480 (meaning the data should be inserted beginning at btye 1,480
- flag = 1 (meaning there is more)

3rd fragment

- 1020 byte (=3980-1480-1480) information field
- identification = 777
- offset = 2,960 (meaning the data should be inserted beginning at byte 2,960)
- flag = 0 (meaning this is the last fragment)

The payload of the datagram is only passed to the transport layer once the IP layer has fully reconstructed the original IP datagram. If one or more of the fragments does not arrive to the destination, the datagram is "lost" and not passed to the transport layer. But, as we learned in the previous chapter, if TCP is being used at the transport layer, then TCP will recover from this loss by having the source retransmit the data in the original datagram.

Fragmentation and reassembly puts an additional burden on Internet routers (the additional effort to create fragments out of a datagram) and on the destination hosts (the additional effort to reassembly fragments). For this reason it is desirable to keep fragmentation to a minimum. This is often done by limiting the TCP and UDP segments to a relatively small size, so that the fragmentation of the corresponding datagram’s is unlikely. Because all data link protocols supported by IP are supposed to have MTUs of at least 576 bytes, fragmentation can be entirely eliminated by using a MSS of 536 bytes, 20 bytes of TCP segment header and 20 bytes of IP datagram header. This is why most TCP segments for bulk data transfer (such as with HTTP) are 512-536 bytes long. (You may have noticed while surfing the Web that 500 or so bytes of data often arrive at a time.)

Following this section we provide a Java applet that generates fragments. You provide the incoming datagram size, the MTU and the incoming datagram identification. It automatically generates the fragments for you.

3-6 ICMP: Internet Control Message Protocol

We conclude this section with a discussion of the Internet Control Message Protocol, ICMP, which is used by hosts, routers, and gateways to communicate network layer information to each other. ICMP is specified in

[RFC 792]. The most typical use of ICMP is for error reporting. For example, when running a Telnet, FTP, or HTTP session, you may have encountered an error message such as "Destination network unreachable." This message had its origins in ICMP. At some point, an IP router was unable to find a path to the host specified in your Telnet, FTP or HTTP application. That router created and sent a type-3 ICMP message to your host indicating the error. Your host received the ICMP message and returned the error code to the TCP code that was attempting to connect to the remote host. TCP in turn returned the error code to your application.

ICMP is often considered part of IP, but architecturally lies just above IP, as ICMP messages are carried inside IP packets. That is, ICMP messages are carried as IP payload, just as TCP or UDP packets are carried at IP payload. Similarly, when an host receives an IP packet with ICMP specified as the upper layer protocol, it demultiplexed the packet to ICMP, just as it would demultiplex a packet to TCP or UDP.

ICMP messages have a type and a code field, and also contain the first 8 bytes of the IP packet that caused the IP message to be generated in the first place (so that the sender can determine which packet is sent that caused the error). Selected ICMP messages are shown below in Figure 3-10. Note that ICMP messages are used not only for signaling error conditions. The well-known ping [ping man page] program uses ICMP. Ping sends an ICMP type 8 code 0 message to the specified host. The destination host, seeing the echo request sends back an type 0 code 0 ICMP echo reply. Another interesting ICMP message is the source quench message. This message is seldom used in practice. Its original purpose was to perform congestion control -- to allow a congested router to send an ICMP source quench message to a host to force that host to reduce its transmission rate. We have seen in Chapter 3 that TCP has its own congestion control mechanism that operates at the transport layer, without the use of network layer support such as the ICMP source quench message.

ICMP type	code	description
0	0	echo reply (to ping)
3	0	destination network unreachable
3	1	destination host unreachable
3	2	destination protocol unreachable
3	3	destination port unreachable
3	6	destination network unknown

3	7	destination host unknown
4	0	source quench (congestion control)
8	0	echo request
9	0	router advertisement
10	0	router discovery
11	0	TTL expired
12	0	IP header bad

Figure 3-10: Selected ICMP messages

In unit 1 we introduced the Trace route program, which enabled you to trace the route from a few given hosts to any host in the world. Interesting enough, Trace route also uses ICMP messages. To determine the names and addresses of the routers between source and destination, Trace route in the source sends a series of ordinary IP datagram's to the destination. The first of these datagram's has a TTL of 1, the second of 2, the third of 3, etc. The source also starts timers for each of the datagram's. When the n-the datagram arrives at the nth router, the n-th router observers that the TTL of the datagram has just expired. According to the rules of the IP protocol, the router discards the datagram (because there may be a routing loop) and sends an ICMP warning message to the source (type 11 code 0). This warning message includes the name of the router and its IP address. When the ICMP message corresponding to the nth datagram arrives at the source, the source obtains the round-trip time from the timer and the name and IP address from the ICMP message. Now that you understand how Trace route works, you may want to go back and play with it some more.

4. Routing in the Internet

Structure

Objectives

- Discuss the Intra-Autonomous System Routing in the Internet;
- Describe internet routing protocols, such OSPF and BGP, and
- Introduce Internet Control Message Protocol (ICMP).
- Comparison in between Inter-AS and Intra-AS Routing Protocols

4-1 Introduction

The Internet consists of interconnected autonomous systems (ASs). An AS typically consists of many networks, where a network (also called an IP network) was defined in the previous section. Recall from lesson 3 that each autonomous system is administered independently. The administrator of an autonomous system chooses the intra-AS routing algorithm for that AS, and is responsible for administering that AS and no others. Datagram's must also be routed among the ASs, and this is the job of inter-AS routing protocols. As discussed in lesson 2, this hierarchical organization of the Internet has permitted the Internet to scale. In this section we examine the intra-AS and inter-AS routing protocols for that are commonly used in the Internet.

4-2 Intra-Autonomous System Routing in the Internet

An intra-AS routing protocol is used to configure and maintain the routing tables within an autonomous system (AS). Once the routing tables are configured, datagram's are routed within the AS as described in the previous section. Inter-AS routing protocols are also known as **interior gateway protocols.** Historically, three routing protocols have been used extensively for routing within an autonomous system in the Internet: **RIP** (the Routing Information Protocol), and **OSPF** (Open Shortest Path First), and **IGRP** (Cisco's propriety Interior Gateway Routing Protocol).

4-2-1 RIP: Routing Information Protocol

The Routing Information Protocol (RIP) was one of the earliest intra-AS Internet routing protocols and is still in widespread use today. It traces its origins and its name to the Xerox Network Systems (XNS) architecture. The widespread deployment of RIP was due in great part to its inclusion in 1982 of the Berkeley Software Distribution (BSD) version of UNIX supporting TCP/IP. RIP version 1 is defined in **[RFC 1058],** with a backwards compatible version 2 defined in [RFC 1723].

RIP is a distance vector protocol that operates in a manner very close to the idealized protocol we examined in further sections. The version of RIP specified in RFC 1058 uses hop count as a cost metric, i.e., each link has a cost of 1, and limits the maximum cost of a path to 15. This limits the use of RIP to autonomous systems that are less than 15 hops in diameter. Recall that in distance vector protocols, neighboring routers exchange routing information with each other. In RIP, the routing tables are exchanged between neighbors every 30 seconds using RIP's. This is done with RIP's so-called **response message**, with each *response* message containing that host's routing table entries for up to 25 destination networks. These response messages containing routing tables are also called **advertisements**.

Let us take a look at a simple example of how RIP advertisements work. Consider the portion of an AS shown in Figure 4-1. In this figure, the rectangles denote routers and the lines connecting the rectangles denote networks. Note that the routers are labeled A, B, etc. and the networks are labeled 1, 10, 20, 30, etc. For visual convenience, some of the routers and networks are not labeled. Dotted lines in the figure indicate that the autonomous system continues on and perhaps loops back. Thus this autonomous system has many more routers and links than are shown in the figure.

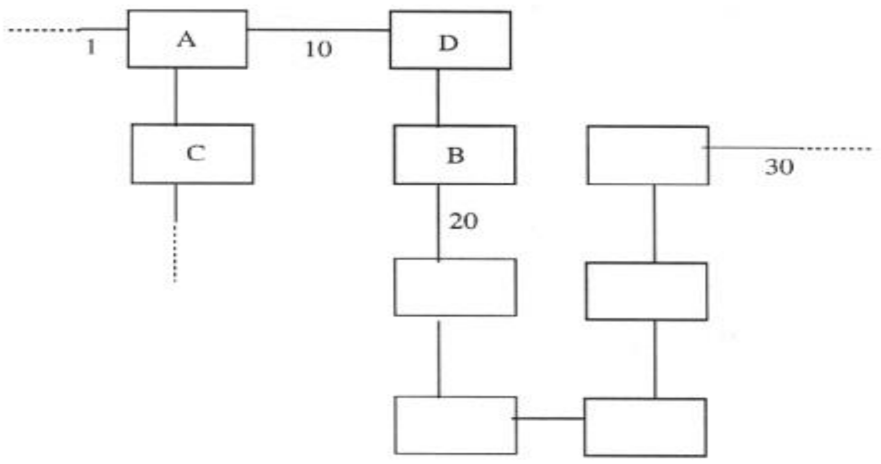


Figure 4-1: A portion of an autonomous system.

Now suppose that the routing table for router D is as shown in Figure 4-2. Note that the routing table has three columns. The first column is for the destination network, the second column indicates the next router along the shortest path to the destination network, and the third column indicates the number of hops (i.e., the number of networks that have to be traversed, including the destination network, to get to the destination network along the shortest path). For this example, the table indicates that to send a datagram from router D to destination network 1, the datagram should be first sent to neighboring router A; moreover, the table indicates that destination network 1 is two hops away along the shortest path. Also note that the table indicates that network 30 is seven hops away via router B. In principle, the routing table should have one row for each network in the AS. It should also have at least one row for networks that are outside of the AS. The tables in Figure 4-2, and the subsequent tables to come, are only partially complete.

destination network	next router	number of hops to destination
1	A	2
20	B	2
30	B	7
10	--	1
....	....	....

Figure 4-2: Routing table in router D before receiving advertisement from router A.

Now suppose that 30 seconds later, router D receives from router A the advertisement shown in Figure 4-3. Note that this advertisement is nothing other but the routing table in router A! This routing table says, in particular, that network 30 is only 4 hops away from router A.

destination network	next router	number of hops to destination
30	C	4
1	--	1
10	--	1
....	....	....

Figure 4-3: Advertisement from router A.

Router D, upon receiving this advertisement, merges the advertisement (Figure 4-3) with the "old" routing table (Figure 4-2). In particular, router D learns that there is now a path through router A to network 30 that is shorter than the path through router B. Thus, router D updates its routing table to account for the "shorter" shortest path, as shown in Figure 4-4. How is it, you might ask, that the shortest path to network 30 became shorter. This is because either this decentralized distance vector algorithm was still in the process of converging, or new links and/or routers were added to the AS, which changed the actual shortest paths in the network.

destination network	**next router**	**number of hops to destination**
1	A	2
20	B	2
30	A	5
....	....	....

Figure 4-4: Routing table in router D after receiving advertisement from router A.

Returning now to the general properties of RIP, if a router does not hear from its neighbor at least once every 180 seconds, that neighbor is considered to be no longer reachable, i.e., either the neighbor has died or the connecting link has gone down. When this happens, RIP modifies its local routing table and then propagates this information by sending advertisements to its neighboring routers (the ones that are still reachable). A router can also request information about its neighbor's cost to a given destination using RIP's *request* message. Routers send RIP request and response messages to each other over UDP using port number 520.The UDP packet is carried between routers in a standard IP packet. The fact that RIP uses a transport layer protocol (UDP) on top of a network layer protocol (IP) to implement network layer functionality (a routing algorithm) may seem rather convoluted (it is!). Looking a little deeper at how RIP is implemented will clear this up.

Figure 4-5 sketches how RIP is typically implemented in a UNIX system, e.g., for example, a UNIX workstation serving as a router. A process called *routed* (pronounced "route dee") executes the RIP protocol, i.e., maintains the routing table and exchanges messages with routed processes running in neighboring routers. Because RIP is implemented as an application-layer process (albeit a very special one that is able to manipulate the routing tables within the UNIX kernel), it can send and receive messages over a standard socket and use a standard transport protocol.

Thus, RIP is an application-layer protocol (see Chapter 2), running over UDP.

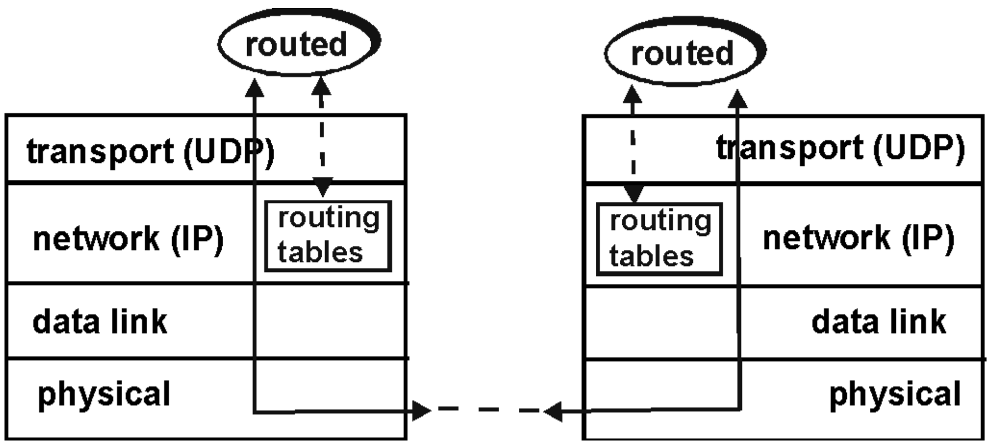


Figure 4-5: Implementation of RIP as the *routed* daemon

Finally, let us take a quick look at a RIP routing table. The RIP routing table below in Figure 4-6 is taken from a UNIX router giroflee.eurecom.fr. If you give a netstat -rn command on a UNIX system, you can view the routing table for that host or router. Performing a netstat on giroflee.eurecom.fr yields the following routing table:

Destination	Gateway	Flags	Ref	Use	Interface
127.0.0.1	127.0.0.1	UH	0	26492	lo0
192.168.2.	192.168.2.5	U	2	13	fa0
193.55.114.	193.55.114.6	U	3	58503	le0
192.168.3.	192.168.3.5	U	2	25	qaa0
224.0.0.0	193.55.114.6	U	3	0	le0
default	193.55.114.129	UG	0	143454	

Figure 4-6 RIP routing table from giroflee.eurecom.fr

The router giroflee is connected to three networks. The second, third and fourth rows in the table tell us that these three networks are attached to giroflee via giroflee's network interfaces fa0, le0 and qaa0. These giroflee interfaces have IP addresses 192.168.2.5, 193.55.114.6 and 192.168.3.5, respectively. To transmit a packet to any host belonging to one of these three networks, giroflee will simply send the outgoing IP datagram over the appropriate interface. Of particular interest to us is the **default route**. Any IP datagram that is not destined for one of the networks explicitly listed in the routing table will be forwarded to the router with IP address 193.55.114.129; this router is reached by sending the datagram over the default network interface. The first entry in the routing table is the so-called loopback interface. When IP sends a datagram to the loopback interface, the packet is simply returned back to IP; this is useful for debugging purposes. The address 224.0.0.0 is a special multicast (Class D) IP address.

4-2-2 OSPF: Open Shortest Path First

Like RIP, the Open Shortest Path First (OSPF) routing is used for intra-AS routing. The "Open" in OSPF indicates that the routing protocol specification is publicly available (e.g., as opposed to Cisco's IGRP protocol). The most recent version of OSPF, version 2, is defined in RFC 2178 a public document.

OSPF was conceived as the successor to RIP and as such has a number of advanced features. At its heart, however, OSPF is a link-state protocol that uses flooding of link state information and a Dijkstra least cost path algorithm. With OSPF, a router constructs a complete topological map (i.e., a directed graph) of the entire autonomous system. The router then locally runs Dijkstra's shortest path algorithm to determine a shortest path tree to all networks with itself as the root node. The router's routing table is then obtained from this shortest path tree. Individual link costs are configured by the network administrator.

Let us now contrast and compare the advertisements sent by RIP and OSPF. With OSPF, a router periodically sends routing information to all other routers in the autonomous system, not just to its neighboring routers. This routing information sent by a router has one entry for each of the router's neighbors; the entry gives the distance (i.e., link state) from the router to the neighbor. On the other hand, a RIP advertisement sent by a router contains information about all the networks in the autonomous system, although this information is only sent to its neighboring routers. In a sense, the advertising techniques of RIP and OSPF are duals of each other.

Some of the advances embodied in OSPF include the following:

- **Security.** All exchanges between OSPF routers (e.g., link state updates) are authenticated. This means that only trusted routers can participate in the OSPF protocol within a domain, thus preventing malicious intruders (or networking students taking their newfound knowledge out for a joyride) from injecting incorrect information into router tables.

- **Multiple same-cost paths.** When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used (i.e., a single path need not be not chosen for carrying all traffic when multiple equal cost paths exist).

- **Different cost metrics for different TOS traffic.** OSPF allows each link to have different costs for different TOS (type of service) IP packets. For example, a high bandwidth satellite link might be configured to have a low cost (and hence be attractive) for non-time critical traffic, but a very high cost metric for delay-sensitive traffic. In essence, OSPF sees different network topologies for different classes of traffic, and hence can compute different routes for each type of traffic.

- **Integrated support for unicast and multicast routing.** Multicast OSPF (RFC 1584) provides simple extensions to OSPF to provide for multicast routing (a topic we cover in more depth in Section 4.8). MOSPF uses the existing OSPF link database and adds a new type of link state advertisement to the existing OSPF link state broadcast mechanism.

- **Support for hierarchy within a single routing domain.** Perhaps the most significant advance in OSPF is the ability to hierarchically structure an autonomous system. Lesson 2 has already looked at the many advantages of hierarchical routing structures. We cover the implementation of OSPF hierarchical routing in the remainder of this section.

As OSPF autonomous system can be configured into "areas." Each area runs its own OSPF link state routing algorithm, with each router in an area broadcasting its link state to all other routers in that area. The internal details of an area thus remain invisible to all routers outside the area. Intra-area routing involves only those routers within the same area.

Within each area, one of more **area border routers** is responsible for routing packets outside the area. Exactly one OSPF area in the AS is configured to be the **backbone** area. The primary role of the backbone area is to route traffic between the other areas in the AS. The backbone always contains all area border routers in the AS and may contain non border routers as well. Inter-area routing within the AS requires that the packet be first routed to an area border router (intra-domain routing), then routed though the backbone to the area border router that is in the destination area, and then routed to the final destination.

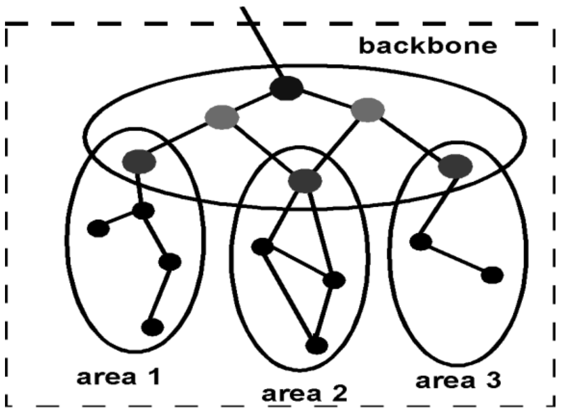


Figure 4-7: Hierarchically structured OSPF AS with four areas.

A diagram of a hierarchically structured OSPF network is shown in Figure 4-7. We can identify four types of OSPF routers in Figure 4-7:

- *Internal routers.* These routers, shown in black, are in non-backbone areas and only perform intra-AS routing.
- *Area border routers.* These routers, shown in blue, belong to both an area and the backbone.
- *Backbone routers (non border routers)*. These routers, shown in gray, perform routing within the backbone but themselves are not area border routers. Within a non-backbone area, internal routers learn of the existence of routes to other areas from information (essentially a link state advertisement, but advertising the cost of a route to another area, rather than a link cost) broadcast within the area by its backbone routers.
- *Boundary routers.* A boundary router, shown in blue, exchanges routing information with routers belonging to other autonomous systems. This router might, for example, use BGP to perform inter-AS routing. It is through such a boundary router that other routers learn about paths to external networks.

4-2-3 IGRP: Internal Gateway Routing Protocol

The Interior Gateway Routing Protocol (IGRP) [Cisco97] is a proprietary routing algorithm developed by Cisco Systems, Inc. in the mid-1980 as a successor for RIP. IGRP is a distance vector protocol. Several cost metrics (including delay, bandwidth, reliability, and load) can be used in making routing decisions, with the weight given to each of the metrics being determined by the network administrator. This ability to use administrator-defined costs in making route selections is an important difference from RIP; we will see shortly that so-called policy-based inter domain Internet routing protocols such as BGP also allow administratively defined routing decisions to be made. Other important differences from RIP include the use of a reliable transport protocol to communicate routing information, the use of update messages that are sent only when routing table costs change (rather than periodically) , and the use of a distributed diffusing update routing algorithm to quickly compute loop free routing paths.

4-3 Inter-Autonomous System Routing: BGP

The Border Gateway Protocol version 4, specified in RFC 1771, is the de facto standard inter domain routing protocol in today's Internet. It is commonly referred to as BGP4 or simply as BGP. As an inter-autonomous system routing protocol, it provides for routing between autonomous systems (that is, administrative domains).

While BGP has the same general flavor as the distance vector protocol that we studied in lesson 2, it is more appropriately characterized as a **path vector protocol.** This is because BGP in a router does not propagate cost information (e.g., number of hops to a destination), but instead propagates path information, such as the sequence of ASs on a route to a destination AS. We will examine the path information in detail shortly. We note though that while this information includes the names of the ASs on a route to the destination, they do *not* contain cost information. Nor does BGP specify how a specific route to a particular destination should be chosen among the routes that have been advertised. That decision is a *policy* decision that is left up to the domain's administrator. Each domain can thus choose its routes according to whatever criteria it chooses (and need not even inform its neighbors of its policy!) allowing a significant degree of autonomy in route selection. In essence, BGP provides the *mechanisms* to distribute path information among the interconnected autonomous systems, but leaves the policy for making the actual route selections up to the network administrator.

Let's begin with a grossly simplified description of how BGP works. This will help us see the forest through the trees. As discussed in lesson 2, as far as BGP is concerned, the whole Internet is a graph of ASs, and each AS is identified by an AS number. At any instant of time, a given AS X may, or may not, know of a path of ASs that leads to a given destination AS Z. As an example, suppose X has listed in its BGP table such a path $XY_1Y_2Y_3Z$ from itself to Z. This means that X knows that it can send datagram's to Z through the ASs X, Y_1, Y_2 and Y_3, Z. When X sends updates to its BGP neighbors (i.e., the neighbors in the graph), X actually sends the entire path information, $XY_1Y_2Y_3Z$, to its neighbors (as well as other paths to other ASs). If, for example, W is a neighbor of X, and W receives an advertisement that includes the path $XY_1Y_2Y_3Z$, then W can list a new entry $WXY_1Y_2Y_3Z$ in its BGP table .However, we should keep in mind that W may decide to *not* create this new entry for one of several reasons. For example, W would not create this entry if W is equal to (say) Y_2, thereby creating an undesirable loop in the routing; or if W already has a path to Z in its tables, and this existing path is preferable (with respect to the metric used by BGP at W) to $WXY_1Y_2Y_3Z$; or, finally, if W has a policy decision to not forward datagram's through (say) Y_2.

In BGP jargon, the immediate neighbors in the graph of ASs are called **peers**. BGP information is proprogated through the network by exchanges of BGP messages between peers. The BGP protocol defines the four types of messages: OPEN UPDATE, NOTIFICATION and KEEPALIVE.

- **OPEN:** BGP peers communicate using the TCP protocol and port number 179. TCP thus provides for reliable and congestion controlled message exchange between peers. In contrast, recall that we earlier saw that two RIP peers, e.g., the *routed's* in Figure 4.5-6 communicate via unreliable UDP. When a BGP gateway wants to first establish contact with a BGP peer (e.g., after the gateway itself or a connecting link has just be booted), an OPEN message is sent to the peer. The OPEN message allows a BGP gateway to identify and authenticate itself, and provide timer information. If the OPEN is acceptable to the peer, it will send back a KEEPALIVE message.

- **UPDATE:** A BGP gateway uses the UPDATE message to advertise a path to a given destination (e.g., $XY_1Y_2Y_3Z$) to the BGP peer. The UPDATE message can also be used to *withdraw* routes that had previously been advertised (that is, to tell a peer that a route that it had previously advertised is no longer a valid route).
- **KEEPALIVE:** This BGP message is used to let a peer know that the sender is alive but that the sender doesn't have other information to send. It also serves as an acknowledgment to a received OPEN message.
- **NOTIFICATION:** This BGP message is used to inform a peer that an error has been detected (e.g., in a previously transmitted BGP message) or that the sender is about to close the BGP session.

Recall from our discussion above that BGP provides mechanisms for distributing path information but does not mandate policies for selecting a route from those available. Within this framework, it is thus possible for an AS such as Hatfield.net to implement a policy such as "traffic from my AS should not cross the AS McCoy.net," since it knows the identities of all AS's on the path. (The Hatfield and the McCoy's are two famous feuding families in the US). But what about a policy that would prevent the McCoy's from sending traffic through the Hatfield's network? The only means for an AS to control the traffic it passes though its AS (known as "transit" traffic - traffic that neither originates in, nor is destined for, the network, but instead is "just passing through") is by controlling the paths that it advertises. For example, if the McCoy's are immediate neighbors of the Hat-fields, the Hat-fields could simply not advertise any routes to the McCoy's that contain the Hatfield network. But restricting transit traffic by controlling an AS's route advertisement can only be partially effective. For example, if the Jones are between the Hatfield's and the McCoy's, and the Hatfield's advertise routes to the Jones' that pass through the Hat-fields, then the Hat-fields can not prevent (using BGP mechanisms) the Jones from advertising these routes to the McCoys.

Very often an AS will have multiple gateway routers that provide connections to other ASs. Even though BGP is an inter-AS protocol, it can still be used inside an AS as a pipe to exchange BGP updates among gateway routers belonging to the same AS. BGP connections inside an AS are called Internal **BGP (IBGP)**, whereas BGP connections between ASs are called External **BGP (EBGP)**.

As noted above, BGP, which is the successor to EGP, is becoming the de facto standard for inter-AS routing for the public Internet. BGP is used for example at the major network access points (NAP's) where major Internet carries connect to each other and exchange traffic. To see the contents of today's (less than four hours out of date) BGP routing table (large!) at one of the major NAP's in the US (which include Chicago and San Francisco), click here.

4-3-1 Why are there Different Inter-AS and Intra-AS Routing Protocols?

Having now studied the details of specific inter-AS and intra-AS routing protocols deployed in today's Internet, let us conclude by considering perhaps the most fundamental question we could ask about these protocols in the first place (hopefully, you have been wondering this all along, and have not lost the forest for the trees!):

Why are different inter-As and intra-AS routing protocols used?
The answer to this question gets at the heart of the differences between the goals of routing within an AS and among ASs:

- **Policy.** Among ASs, policy issues dominate. It may well be important that traffic originating in a given AS specifically not be able to pass through another specific AS. Similarly, a given AS may well want to control what transit traffic it carries between other ASs. We have seen that BGP specifically carries path attributes and provide for controlled distribution of routing information so that such policy-based routing decisions can be made. Within an AS, everything is nominally under the same administrative control, and thus policy issues play a much less important role in choosing routes within the AS.
- **Scale.** The ability of a routing algorithm and its data structures to scale to handle routing to/among large numbers of networks is a critical issue in inter-AS routing. Within an AS, scalability is less of a concern. For one thing, if single administrative domains become too large, it is always possible to divide it into two ASs and perform inter-AS routing between the two new ASs. (Recall that OSPF allows such a hierarchy to be built by splitting an AS into "areas").
- **Performance.** Because inter-AS routing is so policy-oriented, the quality (e.g., performance) of the routes used is often of secondary concern (i.e., a longer or more costly route that satisfies a certain policy criteria may well be taken over a route that is shorter but does not meet that criteria). Indeed, we saw that among ASs, there is not even the notion of preference or costs associated with routes. Within a single AS, however, such policy concerns can be ignored, allowing routing to focus more on the level of performance realized on a route.

5. What's inside a router?

Structure

Objectives

- Discuss internal structure of router;
- Discuss input and out put ports;
- Discuss switching fabrication and queuing concept.

5-1 Introduction

Repeaters and bridges are simple hardware devices capable of executing specific tasks. **Routers** are more sophisticated. They have access to network layer address and contain software that enables them to determine which of several possible paths between those address the best for a particular transmission is. Routers operate in the physical, data link, and network layers of the OSI model. Our study of the network layer so far has focused on network layer service models, the routing algorithms that control the routes taken by packets through the network, and the protocols that embody these routing algorithms. These topics, however, are only part (albeit *important* ones) of what goes on in the network layer. We have yet to consider the **switching function** of a router the actual transfer of datagram's from a router's incoming links to the appropriate outgoing links. Studying just the control and service aspects of the network layer is like studying a company and considering only its management (which controls the company but typically performs very little of the actual "grunt" work that makes a company run!) and its public relations ("Our product will provide this wonderful service to you!"). To fully appreciate what really goes on within a company, one needs to consider the workers. In the network layer, the real work (that is, the reason the network layer exists in the first place) is the forwarding of datagram's. A key component in this forwarding process is the transfer of a datagram from a router's incoming link to an outgoing link. In this section we study how this is accomplished. Our coverage here is necessarily brief, as an entire course would be needed to cover router design in depth. Consequently, we'll make a special effort in this section to provide pointers to material that covers this topic in more depth.

5-2 Router architecture

A high level view of generic router architecture is shown in Figure 5-1. Four components of a router can be identified:

- **Input ports.** The input port performs several functions. It performs the physical layer functionality (shown in light blue in Figure 5-1) of terminating an incoming physical link to a router. It performs the data link layer functionality (shown in dark blue) needed to interoperate with the data link layer functionality (see Chapter 5) on the other side of the incoming link. It also performs a lookup and forwarding function (shown in red) so that a datagram forwarded into the switching fabric of the router emerges at the appropriate output port. Control packets (e.g., packets carrying routing protocol information such as RIP, OSPF or IGMP) are forwarded from the input port to the routing processor. In practice, multiple ports are often gathered together on a single *line card* within a router.
- **Switching fabric.** The switching fabric connects the router's input ports to its output ports. This switching fabric is completely contained with the router - a network inside of a network router!
- **Output ports.** An output port stores the datagram’s that have been forwarded to it through the switching fabric, and then transmits the datagram’s on the outgoing link. The output port thus performs the reverse data link and physical layer functionality as the input port.
- **Routing processor.** The routing processor executes the routing protocols (e.g., the protocols we studied in lesson 3), maintains the routing tables, and performs network management functions, within the router. Since we cover these topics elsewhere in this book, we defer discussion of these topics to elsewhere.

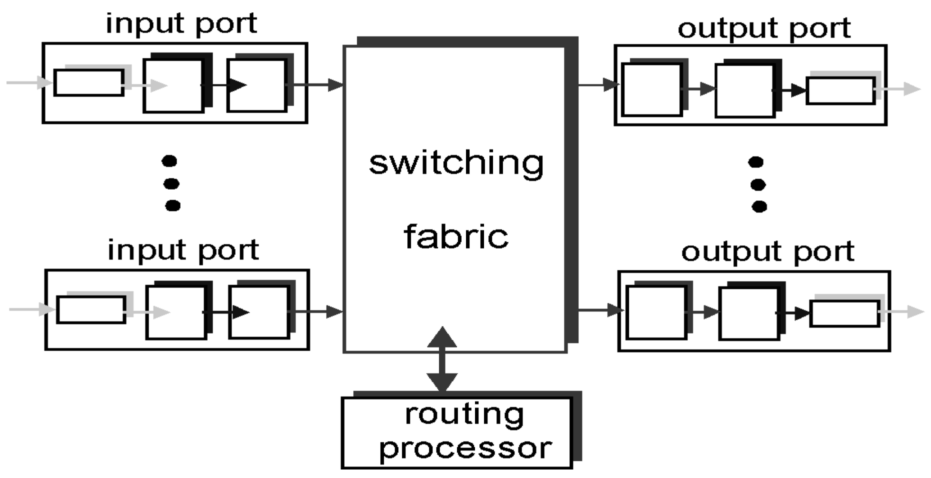


Figure 5-1: Router architecture

. 5-3 Input ports

A more detailed view of input port functionality is given in Figure 5-2. As discussed above, the input port's line termination function and data link processing implement the physical and data link layers associated with an individual input link to the router. The lookup/forwarding function of the input port is central to the switching function of the router. In many routers, it is here that the router determines the output port to which an arriving datagram will be forwarded via the switching fabric. The choice of the output port is made using the information contained in the routing table. Although the routing table is computed by the routing processor, a "shadow copy" of the routing table is typically stored at each input port and updated, as needed, by the routing processor. With local copies of the routing table, the switching decision can be made locally, at each input port, without invoking the centralized routing processor. Such *decentralized* switching avoids creating a forwarding bottleneck at a single point within the router.

In routers with limited processing capabilities at the input port, the input port may simply forward the packet to the centralized routing processor, which will then perform the routing table lookup and forward the packet to the appropriate output port. This is the approach taken when a workstation or server serves as a router; here, the "routing processor" is really just the workstation's CPU and the "input port" is really just a network interface card (e.g., a Ethernet card).

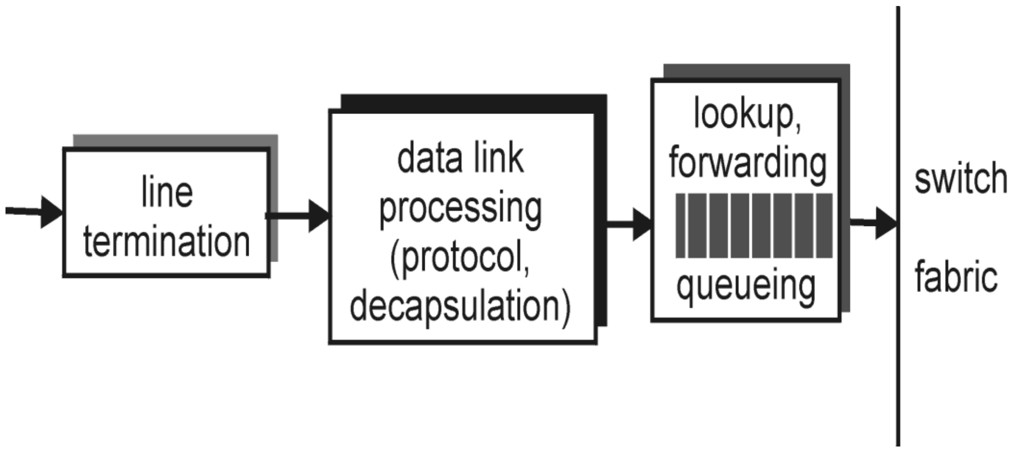


Figure 5-2: Input port processing

Given the existence of a routing table, the routing table lookup is conceptually simple we just search through the routing table, looking for a destination entry that matches the destination address of the datagram, or a default route if the destination entry is missing. In practice, however, life is not so simple. Perhaps the most important complicating factor is that backbone routers must operate at high speeds, being capable of performing millions of lookups per second. Indeed, it is desirable for the input port processing to be able to proceed at **line speed**, i.e., that a lookup can be

NOTES

done in less than the amount of time needed to receive a packet at the input port. In this case, input processing of a received packet can be completed before the next receive operation is complete. To get an idea of the performance requirements for lookup, consider that a so-called OC48 link runs at 2.5Gbps. With 256 byte long packets, this implies a lookup speed of approximately a million lookups per second.

Given the need to operate at today's high link speeds, a linear search through a large routing table is impossible. A more reasonable technique is to store the routing table entries in a tree data structure. Each level in the tree can be thought of as corresponding to a bit in the destination address. To lookup an address, one simply starts at the root node of the tree. If the first address bit is a zero, then the left subtree will contain the routing table entry for destination address; otherwise it will be in the right subtree. The appropriate subtree is then traversed using the remaining address bits if the next address bit is a zero the left subtree of the initial subtree is chosen; otherwise, the right subtree of the initial subtree is chosen. In this manner, one can lookup the routing table entry in N steps, where N is the number of bits in the address. (The reader will note that this is essentially a binary search through an address space of size 2^N.)

But even with N=32 (e.g., a 32-bit IP address) steps, the lookup speed is not fast enough for today's backbone routing requirements. For example, assuming a memory access at each step, less than a million address lookups per sec could be performed with 40 ns memory access times. Several techniques have thus been explored to increase lookup speeds. Content addressable memories (CAMs) allow a 32-bit IP address to be presented to the CAM, which then returns the content of the routing table entry for that address in essentially constant time. The Cisco 8500 series router has a 64K CAM for each input port. Another technique for speeding lookup is to keep recently accessed routing table entries in a cache. Here, the potential concern is the size of the cache. Measurements in suggest that even for an OC-3 speed link, approximately 256,000 source-destination pairs might be seen in one minute in a backbone router. Most recently, even faster data structures, which allow routing table entry to be located in log (N) steps, or which compress routing tables in novel ways have been proposed. A hardware-based approach to lookup that is optimized for the common case that the address being looked up has 24 or less significant bits.

Once the output port for a packet has been determined via the lookup, the packet can be forwarded into the switching fabric. However, as we'll see below, a packet may be temporarily **blocked** from entering the switching fabric (due to the fact that packets from other input ports are currently using the fabric). A blocked packet must thus be queued at the input port and then scheduled to cross the switching fabric at a later point in time. We'll take a closer look at the blocking, queuing and scheduling of packets (at both input ports and output ports) within a router in section 4.6.4 below.

5-4 Switching Fabrics

The switching fabric is at the very heart of a router. It is through this switching that the datagram's are actually moved from an input port to an output port. Switching can be accomplished in a number of ways, as indicated in Figure 5-3:

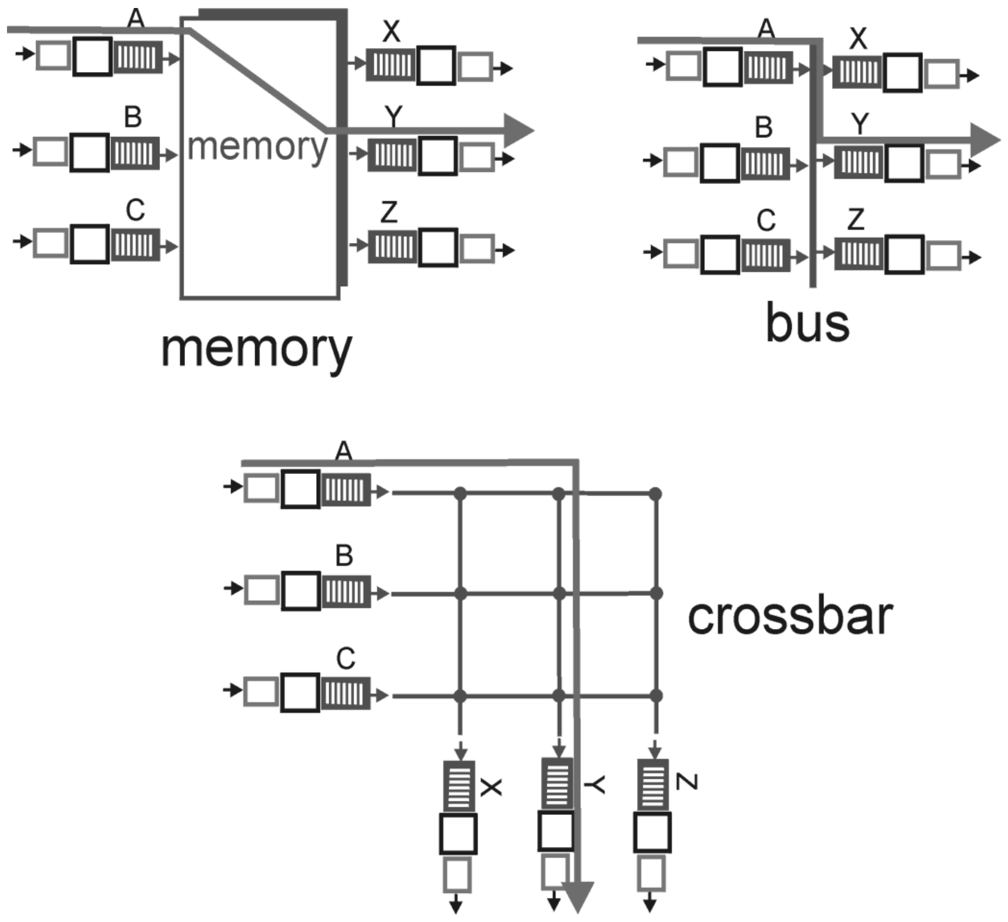


Figure 5-3: Three switching techniques

- **Switching via memory.** The simplest, earliest routers were often traditional computers, with switching between input and output port being done under direct control of the CPU (routing processor). Input and output ports functioned as traditional I/O devices in a traditional operating system. An input port with an arriving datagram first signaled the routing processor via an interrupt. The packet was then copied from the input port into processor memory. The routing processor then extracted the destination address from the header, looked up the appropriate output port in the routing table, and copied the packet to the output port's buffers. Note that if the memory bandwidth is such that B packets/sec can be written into, or read from, memory, then the overall switch throughput (the total rate at which packets are transferred from input ports to output ports) must be less than B/2.

Many modern routers also switch via memory. A major difference from early routers, however, is that the lookup of the destination address and the storing (switching) of the packet into the appropriate memory location is performed by processors on the input line cards. In some ways, routers that switch via memory look very much like shared memory multiprocessors, with the processors on a line line card storing datagram's into the memory of the appropriate output port. Cisco's Catalyst 8500 series switches and Bay Networks Accelar 1200 Series routers switch packets via a shared memory.

- **Switching via a bus.** In this approach, the input ports transfer a datagram directly to the output port over a shared bus, without intervention by the routing processor (Note that when switching via memory, the datagram must also cross the system bus going to/from memory). Although the routing processor is not involved in the bus transfer, since the bus is shared, only one packet at a time can be transferred over the bus at a time. A datagram arriving at an input port and finding the bus busy with the transfer of another datagram is blocked from passing through the switching fabric and queued at the input port. Because every packet must cross the single bus, the switching bandwidth of the router is limited to the bus speed.

  Given that bus bandwidths of over a gigabit per second are possible in today's technology, switching via a bus is often sufficient for routers that operate in access and enterprise networks (e.g., local area and corporate networks). Bus-based switching has been adopted in a number of current router products, including the Cisco 1900, which switches packets over a 1Gbps Packet Exchange Bus. 3Com's Core Builder 5000 systems interconnects ports that reside on different switch modules over its Packet Channel data bus, with a bandwidth of 2 Gbps.

- **Switching via an interconnection network.** One way to overcome the bandwidth limitation of a single, shared bus is to use a more sophisticated interconnection network, such as those that have been used in the past to interconnect processors in multiprocessor computer architectures. A crossbar switch is an interconnection network consisting of 2N busses that connect N input ports to N output ports, as shown in Figure 5-3. A packet arriving at an input port travels along the horizontal bus attached to the input port until it intersects with the vertical bus leading to the desired output port. If the vertical bus leading to the output port is free, the packet is transferred to the output port. If the vertical bus is being used to transfer a packet from another input port to this same output port, the arriving packet is blocked and must be queued at the input port.

Delta and Omega switching fabrics have also been proposed as an interconnection network between input and output ports. See for a survey of switch architectures. Cisco 12000 Family switches use an interconnection network, providing up to 60Gbps through the switching fabric. One current trend in interconnection network design http://ryt.uncoma.edu.ar/material/Kurose/4-network/4_06-inside.htm%5BKeshav%201998%5D is to fragment a variable length IP datagram into fixed length cells, and then tag and switch the fixed length cells through the interconnection network. The cells are then reassembled into the original datagram at the output port. The fixed length cell and internal tag can considerably simplify and speed up the switching of the packet through the interconnection network.

5-5 Output Ports

Output port processing, shown in Figure 5-4, takes the datagram's that have been stored in the output port's memory and transmits them over the outgoing link. The data link protocol processing and line termination are the send-side link- and physical layer functionality that interact with the input port on the other end of the outgoing link, as discussed above in section 5-3. The queuing and buffer management functionality are needed when the switch fabric delivers packets to the output port at a rate that exceeds the output link rate; we'll cover output port queuing below.

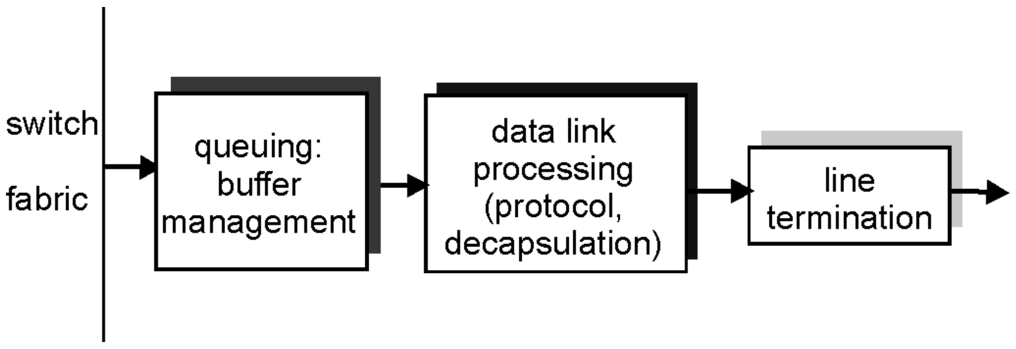

switch
fabric
queuing: buffer management
data link processing (protocol, decapsulation)
line termination

Figure 5-4: Output Port processing

5-6 Where Does Queuing Occur?

Looking at the input and output port functionality and the configurations shown in Figure 5-3, it is evident that packet queues can form at both the input ports *and* the output ports. It is important to consider these queues in a bit more detail, since as these queues grow large, the router's buffer space will eventually be exhausted and **packet loss** will occur. Recall that in our earlier discussions, we said rather vaguely that packets were lost "within the network" or "dropped at a router." It is here, at these queues within a router, where such packets are dropped and lost. The actual location of packet loss (either at the input port queues or the output port queues) will depend on the traffic load, the relative speed of the switching fabric and the line speed, as discussed below.

Suppose that the input line speeds and output line speeds are all identical, and that there are *n* input ports and *n* output ports. If the switching fabric speed is at least *n* times as fast as the input line speed, than no queuing can occur at the input ports. This is because even in the worst case that all *n* input lines are receiving packets, the switch will be able to transfer *n* packets from input port to output port in the time it takes each of the *n* input ports to (simultaneously) receive a *single* packet. But what can happen at the output ports? Let us suppose still that the switching fabric is at least *n* times as fast as the line speeds. In the worst case, the packets arriving at each of the *n* input ports will be destined to the *same* output port. In this case, in the time it takes to receive (or send) a single packet, *n* packets will arrive at this output port. Since the output port can only transmit a single packet in a unit of time (the packet transmission time), the *n* arriving packets will have to queue (wait) for transmission over the outgoing link. '*n*' more packets can then possibly arrive in the time it takes to transmit just one of the *n* packets that had previously been queued. And so on. Eventually, buffers can grow large enough to exhaust the memory space at the output port, in which case packets are dropped.

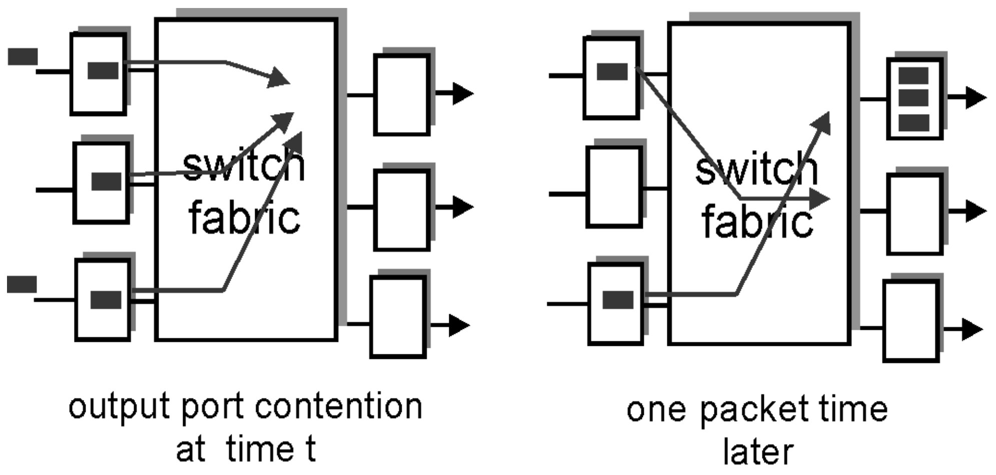


Figure 5-5: output port queuing

Output port queuing is illustrated in Figure 5-5. At time *t*, a packet has arrived at each of the incoming input ports, each destined for the uppermost outgoing port. Assuming identical line speeds and a switch operating at three times the line speed, one time unit later (i.e., in the time needed to receive or send a packet), all three original packets have been transferred to the outgoing port and are queued awaiting transmission. In the next time unit, one of these three packets will have been transmitted over the outgoing link. In our example, two *new* packets have arrived at the incoming side of the switch; one of these packets is destined for this uppermost output port.

A consequence of output port queuing is that a **packet scheduler** at the output port must choose one packet among those queued for transmission. This selection might be done on a simple basis such as first-come-first-served (FCFS) scheduling, or a more sophisticated scheduling discipline such as weighted fair queuing (WFQ), which shares the outgoing link "fairly" among the different end-to-end connections that have packets queued for transmission. Packet scheduling plays a crucial role in providing **quality of service guarantees.**

If the switch fabric is not fast enough (relative to the input line speeds) to transfer *all* arriving packets through the fabric without delay, then packet queuing will also occur at the input ports, as packets must join input port queues to wait their turn to be transferred through the switching fabric to the output port. To illustrate an important consequence of this queuing, consider a crossbar switching fabric and suppose that *(i)* all link speeds are identical *(ii)* that one packet can be transferred from any one input port to a given output port in the same amount of time it takes for packet to be received on an input link and *(iii)* packet are moved from a given input queue to their desired output queue in a FCFS manner. Multiple packets can be transferred in parallel, as long as their output ports are different. However, if two packets at the front of two input queues are destined to the same output queue, then one of the packets be blocked and must wait at the input queue - the switching fabric can only transfer one packet to a given output port at a time.

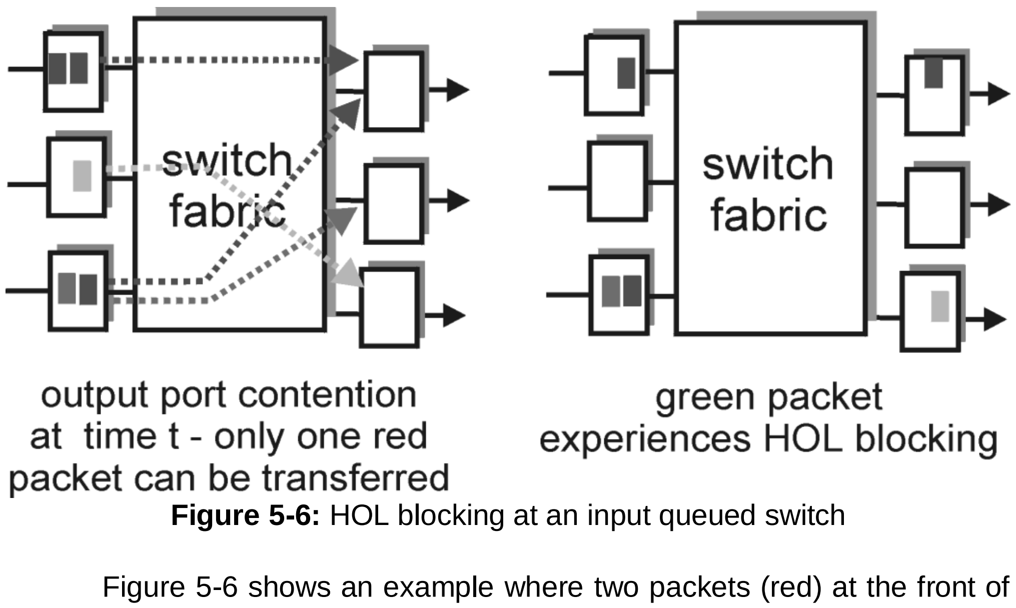

switch fabric
output port contention at time t - only one red packet can be transferred
switch fabric
green packet experiences HOL blocking

Figure 5-6: HOL blocking at an input queued switch

Figure 5-6 shows an example where two packets (red) at the front of their input queues are destined for the same upper right output port. Suppose that the switch fabric chooses to transfer the packet from the front of the upper left queue. In this case, the red packet in the lower left queue must wait. But not only must this red packet wait, but so too must the green packet that is queued behind that packet in the lower left queue, even though there is *no* contention for the middle right output port (the destination for the green packet). This phenomenon is known as **head-of-the-line (HOL) blocking** in an input-queued switch a queued packet in an input queue must wait for transfer through the fabric (even though its output port is free) due to the blocking of another packet at the head-of-the-line. Shows that due to HOL blocking, the input queue will grow to unbounded length (informally, this is equivalent to saying that significant packet loss will occur) as soon as packet arrival rate on the input links reaches only 58% of their capacity.

6. IPv6

Structure

Objectives

- Discuss format of IPv6;
- Discuss about new ICMP for IPv6;
- Describe about the transitioning from IPv4 to IPv6

6-1 Introduction

In the early 1990's the Internet Engineering Task force began an effort to develop a successor to the IPv4 protocol. A prime motivation for this effort was the realization that the 32-bit IP address space was beginning to be used up, with new networks and IP nodes being attached to the Internet (and being allocated unique IP addresses) at a breathtaking rate. To respond to this need of a large IP address space, a new IP protocol, IPv6, was developed. The designers of IPv6 also took this opportunity to tweak and augment other aspects of IPv4, based on the accumulated operational experience with IPv4.

The point in time when IPv4 addresses would have been completely allocated (and hence no new networks could have attached to the Internet) was the subject of considerable debate. Based on current trends in address allocation, the estimates of the two leaders of the IETF's Address Lifetime Expectations working group were that addresses would become exhausted in 2008 and 2018 respectively. In 1996, the American Registry for Internet Number (ARIN) reported that all of the IPv4 class A addresses have been assigned, 62% of the class B addresses have been assigned, and 37% of the class C addresses have been assigned. While these estimates and numbers suggested that a considerable amount of time might be left until the IPv4 address space became exhausted, it was realized that considerable time would be needed to deploy a new technology on such an extensive scale, and so the "Next Generation IP" (IPng) effort was begun. An excellent on-line source of information about IPv6 is The IP Next Generation Homepage.

6-2 IPv6 Packet Format

The format of the IPv6 packet is shown in Figure 6-1. The most important changes introduced in IPv6 are evident in the packet format:

- *Expanded addressing capabilities.* IPv6 increases the size of the IP address from 32 to 128 bits. This insures that the world won't run out of IP addresses. Now, every grain of sand on the planet can be IP-addressable. In addition, the address space contains new hierarchical structure, allocating portions of the enlarged address space to geographical regions. In addition to unicast and multicast addresses, a new type of address, called an **anycast address**, has also been introduced, which allows a packet addressed to an anycase address to be delivered to any one of a group of hosts. This feature could be used, for example, to send an HTTP GET to the nearest of a number of mirror sites that contain a given document).

- *A streamlined 40 byte header.* As discussed below, a number of IPv4 fields have been dropped or made optional. The resulting 40-byte fixed-length header allows for faster processing of the IP packet. A new encoding of options allows for more flexible options processing.
- *Flow labeling and priority.* IPv6 has an elusive definition of a "**flow**" and state this allows "labeling of packets belonging to particular flows for which the sender requests special handling, such as a non-default quality of service or real-time service." For example, audio and video transmission might likely be treated as a flow. On the other hand, the more traditional applications, such as file transfer and email might not be treated as flows. It is possible that the traffic carried by a high-priority user (e.g., someone paying for better service for their traffic) might also be treated as a flow. What is clear, however, is that the designers of IPv6 foresee the eventual need to be able to differentiate among the "flows," even if the exact meaning of a flow has not yet been determined. The IPv6 header also has a 4-bit **priority** field. This field, as the TOS field in IPv4, can be used to give priority to certain packets within a flow, or it can be used to give priority to datagram's from certain applications (e.g., ICMP packets) over packets from other applications (e.g., network news).

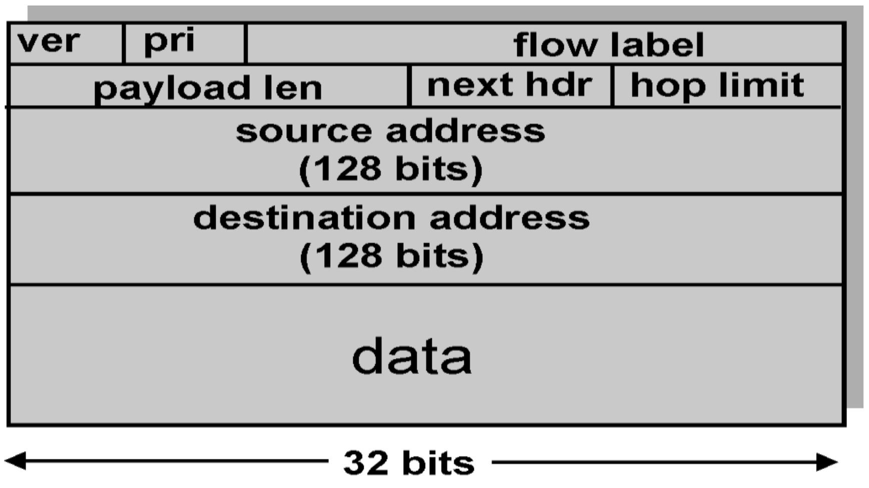
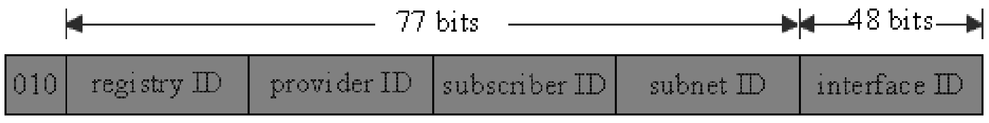


Figure 6-1: IPv6 packet format

The IPv6 packet format is shown in Figure 6-1. As noted above, a comparison of Figure 6-1 with Figure 4.4-8 reveals the simpler, more streamlined structure of the IPv6 packet. The following packet fields are defined in IPv6:

- **Version.** This four bit field identifies the IP version number. Not surprisingly, IPv6 carries a value of "6" in this field. Note that putting a "4" in this field does not create a valid IPv4 packet (if it did, life would be a lot simpler -- see the discussion below regarding the transition from IPv4 to IPv6.
- **Priority.** This four bit field is similar in spirit to the ToS field we saw in IP version 4. States that values 0 through 7 are to be used for priority among traffic that is congestion-controlled (i.e., for which the source will back off on detection of congestion), while values 8 through 15 are used for non-congestion controlled traffic, such as constant bit rate real-time traffic.
- **Flow label**. As discussed above, this field is used to identify a "flow" of packets.
- **Payload length.** This 16-bit value is treated as an unsigned integer given the number of bytes in the IPv6 packet following the fixed length, 40 byte packet header.
- **Next header.** This field identifies the protocol to which the contents (data field) of this packet will be delivered (e.g., to TCP or UDP). The field uses the same values as the Protocol field in the IPv4 header.
- **Hop limit.** The contents of this field are decremented by one by each router that forwards the packet. If the hop limit count reaches zero, the packet is discarded.
- **Source and destination address.** An IP v6 address has the following structure:



- **Data.** This is the payload portion of the IPv6 packet. When the packet reaches its destination, the payload will be removed from the IP packet and passed on to the protocol specified in the next header field.

The discussion above identified the purpose of the fields that *are* included in the IPv6 packet. Comparing the IPv6 packet format in Figure 6-1 with the IPv4 packet format that we saw earlier in Figure 6-8, we notice that several fields appearing in the IPv4 packet are no longer present in the IPv6 packet:

- **Fragmentation/Reassembly.** IPv6 does not provide for fragmentation and reassembly. If an IPv6 packet received by a router is too large to be forwarded over the outgoing link, the router simply drops the packet and sends a "Packet Too Big" ICMP error message (see below) back to the sender. The sender can then resend the data, using a smaller IP packet size. Fragmentation and reassembly is a time-consuming operating; removing this functionality from the routers and placing it squarely in the end systems considerably speeds up IP forwarding within the network.

- **Checksum.** Because the transport layer (e.g, TCP and UDP) and data link (e.g., Ethernet) protocols in the Internet layers perform checksumming, the designers of IP probably felt that this functionality was sufficiently redundant in the network layer that it could be removed. Once again, fast processing of IP packets was a central concern. Recall from our discussion of IPv4 in lesson 3, that since the IPv4 header contains a TTL field (similar to the hop limit field in IPv6), the IPv4 header checksum needed to be recomputed at every router. As with fragmentation and reassembly, this too was a costly operation in IPv4.
- **Options.** An options field is no longer a part of the standard IP header. However, it has not gone away. Instead, the options field is one of the possible "next headers" pointed to from within the IPv6 header. That is, just as TCP or UDP protocol headers can be the next header within an IP packet, so too can an options field. The removal of the options filed results in a fixed length, 40 byte IP header.

6-3 A New ICMP for IPv6

Recall from our discussion in lesson 2, that the ICMP protocol is used by IP nodes to report error conditions and provide limited information (e.g., the echo reply to a ping message) to an end system. In addition to reorganizing the existing ICMP type and code definitions, ICMPv6 also added new types and codes required by the new IPv6 functionality. These include the "Packet Too Big" type, and an "unrecognized IPv6 options" error code. In addition, ICMPv6 subsumes the functionality of the Internet Group Management Protocol (IGMP), which is used to manage a host's joining and leaving of so-called multicast groups, was previously a separate protocol from ICMP in IPv4.

6-4 Transitioning from IPv4 to IPv6

Now that we have seen the technical details of IPv6, let us consider a very practical matter: how will the public Internet, which is based on IPv4, be transitioned to IPv6? The problem is that while new IPv6-capable systems can be made "backwards compatible", i.e., can send, route, and receive IPv4 packets, already deployed IPv4-capable systems are not capable of handling IPv6 packets. Several options are possible.

One option would be to declare a "flag day" - a given time and date when all Internet machines would be turned off, be upgraded from IPv4 to

IPv6. The last major technology transition (from using NCP to using TCP for reliable transport service) occurred almost 20 years ago. When the Internet was tiny and still being administered by a small number of "wizards," it was realized the flag day was not possible. A flag day involving hundreds of millions of machines and millions of network administrators and users is even more unthinkable today. [RFC 1993] describes two approaches (which can be used either alone or together) for gradually integrating IPv6 hosts and routers into an IPv4 world (with the long term goal, of course, of having all IPv4 nodes eventually transition to IPv6).

Probably the most straightforward way to introduce IPv6-capable nodes is a **dual stack** approach, where IPv6 nodes also have a complete IPv4 implementation as well. Such a node, referred to as IPv6/IPv4 node in [RFC 1993], the ability to send and receive both IPv4 and IPv6 packets. When interoperating with an IPv4 node, an IPv6/IPv4 node can use IPv4 packets; when interoperating with an IPv6 node, it can speak IPv6. IPv6/IPv4 nodes must have both IPv6 and IPv4 addresses. They must furthermore be able to determine whether another node is IPv6-capable or IPv4-only. This problem can be solved using the DNS (see Chapter 2), which can return an IPv6 address if the node name being resolved is IPv6 capable, or otherwise return an IPv4 address. Of course, if the node issuing the DNS request in only IPv4 capable, the DNS returns only an IPv4 address.

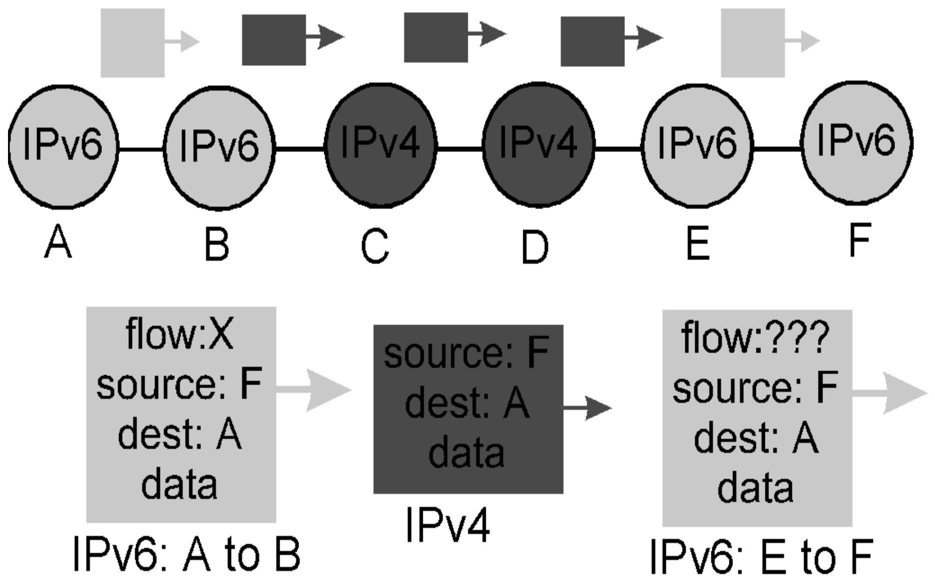


Figure 6-3: A dual stack approach

In the dual stack approach, if either the sender of the receiver is only

IPv4-capable, IPv4 packets must be used. As a result, it is possible that two IPv6-capable nodes can end, in essence, sending IPv4 packets to each other. This is illustrated in Figure 6-3. Suppose node A is IPv6 capable and wants to send an IP packet to node E, which is also IPv6-capable. Nodes A and B can exchange an IPv6 packet. However, node B must create an IPv4 packet to send to C. Certainly, the data field of the IPv6 packet can be copied into the data field of the IPv4 packet and appropriate address mapping can be done. However, in performing the conversion from IPv6 to IPv4, there will be IPv6-specific fields in the IPv6 packet (e.g., the flow identifier field) that have no counterpart in IPv4. The information is these fields will be lost. Thus, even though E and F can exchange IPv6 packets, the arriving IPv4 packets at E from D do not contain all of the fields that were in the original IPv6 packet sent from A. .

An alternative to the dual stack approach, also discussed in [RFC 1993], is known as **tunneling**. Tunneling can solve the problem noted above, allowing, for example, E to receive the IPv6 packet originated by A. The basic idea behind tunneling is the following. Suppose two IPv6 nodes (e.g, B and E in Figure 6-3) want to interoperate using IPv6 packets, but are connected to each other by intervening IPv4 routers. We refer to the intervening set of IPv4 routers between two IPv6 routers as a **tunnel,** as illustrated in Figure 6-4. With tunneling, the IPv6 node on the sending side of the tunnel (e.g., B) takes the *entire* IPv6 packet, and puts it in the data (payload) field of an IPv4 packet. This IPv4 packet is then addressed to the IPv6 node on the receiving side of the tunnel (e.g., E) and sent to the first node in the tunnel (e.g., C). The intervening IPv4 routers in the tunnel route this IPv4 packet amongst themselves, just as they would any other packet, blissfully unaware that the IPv4 packet itself contains a complete IPv6 packet. The IPv6 node on the receiving side of the tunnel eventually receives the IPv4 packet (it is the destination of the IPv4 packet!), determines that the IPv4 packet contains an IPv6 packet, extracts the IPv6 packet and then routes the IPv6 packet exactly as it would if it had received the IPv6 packet from a directly-connected IPv6 neighbor.

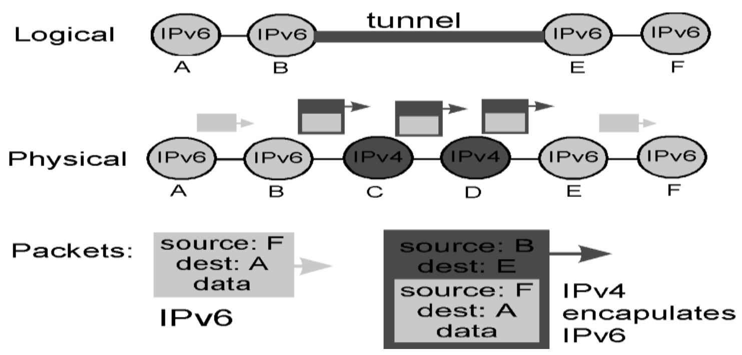


Figure 6-4: Tunneling

We end this section by mentioning that there is currently some doubt about whether IPv6 will make significant inroads into the Internet in the near future (2000-2002) or even ever at all [Garber 1999]. Indeed, at the time of this writing, a number of North American ISPs have said they don't plan to buy IPv6-enabled networking equipment. These ISPs say that there is little customer demand for IPv6's capabilities when IPv4, with some patches (such as network address translator boxes), is working well enough. On the other hand, there appears to be more interest in IPv6 in Europe and Asia. Thus the fate of IPv6 remains an open question.

One important lesson that we can learn from the IPv6 experience is that it is enormously difficult to change network-layer protocols. Since the early 1990s, numerous new network-layer protocols have been trumpeted as the next major revolution for the Internet, but most of these protocols have had minor (if any) penetration to date. These protocols include IPv6, multicast protocols (Section 4.8), and resource reservation protocols (Section 6.9). Indeed, introducing new protocols into the network layer is like replacing the foundation of a house - it is difficult to do without tearing the whole house down or at least temporarily relocated the house's residents. On the other hand, the Internet has witnessed rapid deployment of new protocols at the application layer. The classic example, of course, is HTTP and the Web; other examples include audio and video streaming and chat. Introducing new application layer protocols is like adding a new layer of paint to a house it is relatively easy to do, and if you choose an attractive color, others in the neighborhood will copy you. In summary, in the future we can expect to see changes in the Internet's network layer, but these changes will likely occur on a time scale that is much slower than the changes that will occur at the application layer.

Summary

In this unit we began our journey into the network core. We learned that the network layer requires the coordination of each and every host and router in the network. Because of this, network layer protocols are among the most challenging in the protocol stack.

We learned that one of the biggest challenges in the network layer is routing packets through a network of millions of hosts and routers. We saw that this scaling problem is solved by partitioning large networks into independent administrative domains, which are called autonomous systems (ASs) in the jargon of computer networking. Each AS independently routes its packets through the AS, just as each country independently routes its postal mail through the country. In the Internet, the two most popular protocols for intra-AS routing are currently RIP and OSPF. To route packets among ASs, an inter-AS routing protocol is needed. The dominant inter-AS protocol today is BGP4.

Performing routing on two levels one level for within each of the ASs and another level for among the ASs is referred to as hierarchical routing.

We saw that the scaling problem of routing packets through millions of hosts and routers is largely solved by a hierarchical organization of the network. This is a general principle we should keep in mind when designing protocols, particularly for network-layer protocols: scaling problems can often be solved by hierarchical organizations. It is interesting to note that this principle has been applied throughout the ages to many of other disciplines besides computer networking, including corporate, government, religious and military organizations.

In lesson 1 we also learned about a second scaling issue: For large computer networks, a router may need to process millions of flows of packets between different source-destination pairs at the same time. To permit a router to process such a large number of flows, network designers have learned over the years that the router's tasks should be as simple as possible. Many measures can be taken to make the router's job easier, including using a datagram network layer rather than virtual-circuit network layer, using a streamlined and fixed-sized header (as in IPv6) , eliminating fragmentation (also done in IPv6) and providing the one and only best-effort service. Perhaps the most important trick here is to *not* keep track of individual flows, but instead base routing decisions solely on a hierarchical-structured destination addresses in the packets. It is interesting to note that the postal service has been using this same trick for many years.

In lesson 2 & 3 we also looked at the underlying principles of routing algorithms. We learned that designers of routing algorithms abstract the computer network to a graph with nodes and links. With this abstraction, we can exploit the rich theory of shortest-path routing in graphs, which has been developed over the past 40 years in the operations research and algorithms communities. We saw that are two broad approaches, a centralized approach in which each node obtains a complete map of the network and applies independently a shortest-path routing algorithm; and a decentralized approach, in which individual nodes only have a partial picture of the entire network, yet the nodes work together to deliver packets along the shortest routes. Routing algorithms in computer networks had been an active research area for many years, and will undoubtedly remain so.

At the end of unit we examined two advanced subjects, reflecting current trends in computer networking and the Internet. The first subject is IPv6, which provides a streamlined network layer and resolves the IPv4 address space problem. The second subject is multicast routing, which can potentially save tremendous amounts of bandwidth, router and server resources in a computer networking. It will be interesting to see how the deployment of IPv6 and multicast routing protocols plays out over the next decade of computer networking.

Having computed our study of the network layer, our journey now takes us one further step down the protocol stack, namely, to the link layer. Like the network layer, the link layer is also part of the network core. But we will see in the next chapter that the link layer has the much more localized task of moving packets between nodes on the same link or LAN. Although this task may appear on the surface trivial compared to that of network layer's tasks, we will see that the link layer involves a lot of important and fascinating issues that can keep us busy for a long time.

Exercise

1. What are the two main functions of a datagram-based network layer? What additional functions does a VC based network layer have?
2. List and describe the ATM network service models.
3. Compare and contrast link state and distance vector routing algorithms.
4. Discuss how a hierarchical organization of the Internet has helped to scale to millions of users.
5. Is it necessary that every autonomous system use the same intra AS routing algorithm? Why or why not?
6. What is the 32-bit binary equivalent of the IP address 223.1.3.27?
7. Consider a LAN to which ten host interfaces and three router interfaces are attached. Suppose the LAN uses class C addresses. The IP addresses for the 13 interfaces will be identical in which of the first 32 bits?
8. Consider a router with three interfaces. Suppose all three interfaces use class C addresses. Will the IP addresses of the three interfaces necessarily have the same first eight bits?
9. Suppose there are three routers between a source host and a destination host. Ignoring fragmentation, an IP segment sent from the source host to the destination host will travel over how many interfaces? How many forwarding tables will be indexed to move the datagram from the source to the destination?
10. Suppose an application generates chunks of 40 bytes of data every 20 msec, and each chunk gets encapsulated in a TCP segment and then an IP datagram. What percentage of each datagram will be overhead, and what percentage will be application data?
11. Consider sending a 3000-byte datagram into a link that has an MTU of 500 bytes. Suppose the original datagram is stamped with the identification number 422. How many fragments are generated? What are their characteristics?
12. Compare and contrast the advertisements used by RIP and OSPF.
13. Fill in the blank: RIP advertisements typically announce the number of hops to various destinations. BGP updates, on the other hand, announce the to he various destinations.
14. Why are different inter AS and intra AS protocols used in the internet?
15. Describe three different types of switching fabrics commonly used in packet switches.

16. Why are buffers needed at the output ports of a switch? Why are buffers needed at the input port of a switch?
17. Compare and the IPv4 and the IPv6 header fields. Do they have any fields in common?
18. It has been said that IPv6 tunnels through IPv4 routers; IPv6 treats the IPv4 tunnels as link layer protocols. Do you agree with this statement? Why or why not?
19. What is an important difference between implementing the multicast abstraction via multiple uncast, and a single network (router) supported multicast group?
20. True or false: When a host joins a multicast group, it must change its IP address to be that of the multicast group it is joining.

UNIT – IV

1. The Data Link Layer: Introduction, Services

Structure

Objectives

- The functionality of the Data Link Layer;
- Discuss services of the data link layer;
- Discuss about adapters communicating.

1.1 Introduction

In the previous unit we learned that the network layer provides a communication service between two hosts. As shown in Figure 1-1, this communication path starts at the source host, passes through a series of routers, and ends at the destination host. We'll find it convenient here to refer to the hosts and the routers simply as **nodes (**since, as we'll see shortly, we will not be particularly concerned whether a node is a router or a host), and to the communication channels that connect adjacent nodes along the communication path as **links**. In order to move a datagram from source host to destination host, the datagram must be moved over each of the *individual links* in the path. In this chapter, we focus on the **data link layer,** which is responsible for transferring a datagram across an individual link. We'll see that many different types of link-level technology can be used to connect two nodes. In further sections, we'll examine specific link-level architectures and protocols in more detail.

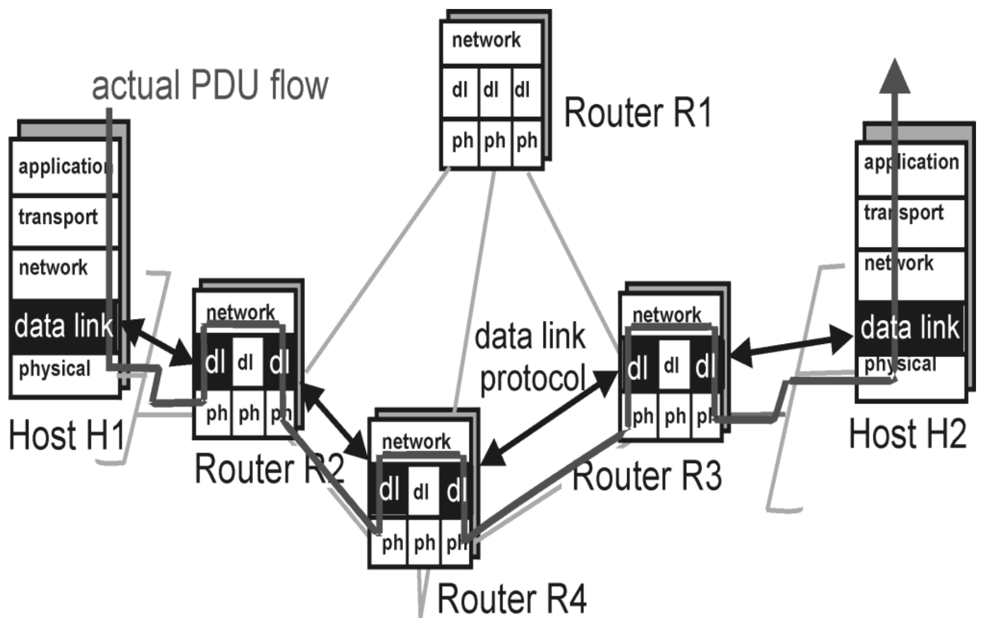


Figure 1-1: The Data Link Layer

1.2 The Services Provided by the Link Layer

A link-layer protocol is used to move a datagram over an individual link. The **link-layer protocol** defines the format of the packets exchanged between the nodes at the ends of the link, as well as the actions taken by these nodes when sending and receiving packets. Recall from Chapter 1 that the packets exchanged by a link-layer protocol are called **frames**, and that each link-layer frame typically encapsulates one network-layer datagram. As we shall see shortly, the actions taken by a link-layer protocol when sending and receiving frames include error detection, retransmission, flow control and random access. Examples of link-layer protocols include Ethernet, token ring, FDDI, and PPP; in some contexts, ATM and frame relay can be considered link-layer protocols as well. We will cover these protocols in detail in the latter half of this chapter.

Whereas the network layer has the end-to-end job of moving transport-layer segments from the source host to the destination host, a link-layer protocol has the node-to-node job of moving a network-layer datagram over a *single link* in the path. An important characteristic of the link layer is that a datagram may be handled by different link-layer protocols on the different links in the path. For example, a datagram may be handled by Ethernet on the first link, PPP on the last link, and frame relay on all intermediate links. It is important to note that the services provided by the different link-layer protocols may be different. For example, a link-layer protocol may or may not provide reliable delivery. Thus, the network layer must be able to accomplish its end-to-end job in the face of a varying set of individual link-layer services.

In order to gain insight to the link layer and how it relates to the network layer, let's consider a transportation analogy. Consider a travel agent who is planning for a tourist traveling from Princeton, New Jersey to Lausanne, Switzerland. Suppose the travel agent decides that it is most convenient for the tourist to take a limousine from Princeton to JFK airport, then a plane from JFK airport to Geneva airport, and finally a train from Geneva to Lausanne's train station. (There is a train station at Geneva's airport.) Once the travel agent makes the three reservations, it is the responsibility of the Princeton limousine company to get the tourist from Princeton to JFK; it is the responsibility of the airline company to get the tourist from JFK to Geneva; and it is responsibility of the Swiss train service to get the tourist from the Geneva to Lausanne. Each of the three segments of the trip is "direct" between two "adjacent" locations. Note that the three transportation segments are managed by different companies and use entirely different transportation modes (limousine, plane and train). Although the transportation modes are different, they each provide the basic service of moving passengers from one location to an adjacent location. This *service* is used by the travel agent to plan the tourist's trip. In this transportation analogy, the tourist is analogous to a datagram, each transportation segment is analogous to a communication link, the transportation mode is analogous to the link-layer protocol, and the travel agent who plans the trip is analogous to a routing protocol.

The basic service of the link layer is to "move" a datagram from one node to an adjacent node over a single communication link. But the details of the link-layer service depend on the specific link-layer protocol that is employed over the link. Possible services that can be offered by a link-layer protocol include:

- **Framing and link access:** Almost all link-layer protocols encapsulate each network-layer datagram within a link-layer frame before transmission onto the link. A frame consists of a data field, in which the network-layer datagram is inserted, and a number of header fields. (A frame may also include trailer fields; however, we will refer to both header and trailer fields as header fields.) A data link protocol specifies the structure of the frame, as well as a channel access protocol that specifies the rules by which a frame is transmitted onto the link. For point-to-point links that have a single sender on one end of the link and a single receiver at the other end of the link, the link access protocol is simple (or non-existent) - the sender can send a frame whenever the link is idle. The more interesting case is when multiple nodes share a single broadcast link - the so-called multiple access problems. Here, the channel access protocol serves to coordinate the frame transmissions of the many nodes; we cover multiple access protocols. We'll see several different frame formats when we examine specific link-layer protocols. We'll see that frame headers also often include fields for a node's so-called **physical address**, which is completely *distinct* from the node's network layer (e.g., IP) address.

- **Reliable delivery:** If a link-layer protocol provides the reliable-delivery service, then it guarantees to move each network-layer datagram across the link without error. Recall that transport-layer protocols (such as TCP) may also provide a reliable-delivery service. Similar to a transport-layer reliable-delivery service, a link-layer reliable-delivery service is achieved with acknowledgments and retransmissions. A link-layer reliable-delivery service is often used for links that are prone to high error rates, such as a wireless link, with the goal of correcting an error locally, on the link at which the error occurs, rather than forcing an end-to-end retransmission of the data by transport or application-layer protocol. However, link-layer reliable delivery is often considered to be unnecessary overhead for low bit-error links, including fiber, coax and many twisted-pair copper links. For this reason, many of the most popular link-layer protocols do not provide a reliable-delivery service.

- **Flow control:** The nodes on each side of a link have a limited amount of packet buffering capacity. This is a potential problem, as a receiving node may receive frames at a rate faster than it can process the frames (over some time interval). Without flow control, the receiver's buffer can overflow and frames can get lost. Similar to the transport layer, a link-layer protocol can provide flow control in order to prevent the sending node on one side of a link from overwhelming the receiving node on the other side of the link.

- **Error detection:** A node's receiver can incorrectly decide that a bit in a frame to be a zero when it was transmitted as a one (and vice versa). These errors are introduced by signal attenuation and electromagnetic noise. Because there is no need to forward a datagram that has an error, many link-layer protocols provide a mechanism for a node to detect the presence of one or more errors. This is done by having the transmitting node set error detection bits in the frame, and having the receiving node perform an error check. Error detection is a very common service among link-layer protocols. The transport layer and network layers in the Internet also provide a limited form of error detection. Error detection in the link layer is usually more sophisticated and implemented in hardware.

- **Error correction:** Error correction is similar to error detection, except that a receiver can not only detect whether errors have been introduced in the frame but can also determine exactly where in the frame the errors have occurred (and hence correct these errors). Some protocols (such as ATM) provide link-layer error correction for the packet header rather than for the entire packet.

- **Half-Duplex and Full-Duplex:** With full-duplex transmission, both nodes at the ends of a link may transmit packets at the same time. With half-duplex transmission, a node cannot both transmit and receive at the same time.

As noted above, many of the services provided by the link layer have strong parallels with services provided at the transport layer. For example, both the link layer and the transport layer can provide reliable delivery. Although the mechanisms used to provide reliable delivery in the two layers are similar, the two reliable delivery services are not the same. A transport protocol provides reliable delivery between two processes on an end-to-end basis; a reliable link-layer protocol provides the reliable-delivery service between two nodes connected by a single link. Similarly, both link-layer and transport-layer protocols can provide flow control and error detection; again, flow control in a transport-layer protocol is provided on an end-to-end basis, whereas it is provided in a link-layer protocol on a node-to-adjacent-node basis.

1-3 Adapters Communicating

For a given communication link, the link-layer protocol is for the most part implemented in a pair of **adapters**. An adapter is a board (or a PCMCIA card) that typically contains RAM, DSP chips, a host bus interface and a link interface. Adapters are also commonly known as *network interface cards* or *NICs*. As shown in Figure 1-2, the network layer in the transmitting node (i.e., a host or router) passes a network-layer datagram to the adapter that handles the sending side of the communication link. The adapter encapsulates the datagram in a frame and then transmits the frame into the communication link. At the other side, the receiving adapter receives the entire frame, extracts the network-layer datagram, and passes it to the network layer. If the link-layer protocol provides error detection, then it is the sending adapter that sets the error detection bits and it is the receiving adapter that performs the error checking. If the link-layer protocol provides reliable delivery, then the mechanisms for reliable delivery (e.g., sequence numbers, timers and acknowledgments) are entirely implemented in the adapters. If the link-layer protocol provides random access, then the random access protocol is entirely implemented in the adapters.

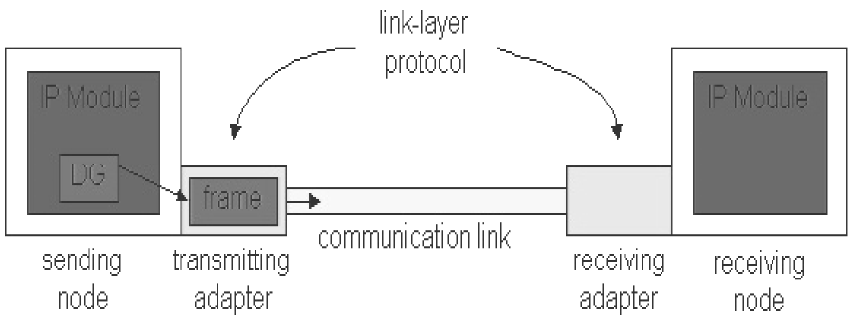


Figure 1-2: The link-layer protocol for a communication link is implemented in the adapters at the two ends of the link. DG abbreviates "datagram".

A computer in itself, an adapter is a semi-autonomous unit. For example, an adapter can receive a frame, determine if a frame is in error and discard the frame without notifying its "parent" node. An adapter that receives a frame only interrupts its parent node when it wants to pass a network-layer datagram up the protocol stack. Similarly, when a node passes a datagram down the protocol stack to an adapter, the node fully delegates to the adapter the task of transmitting the datagram across that link. On the other hand, an adapter is not a completely autonomous unit. Although we have shown the adapter as a separate "box" in Figure 1-3, the adapter is typically housed in the same physical box as rest of the node, shares power and busses with the rest of the node, and is ultimately under the control of the node.

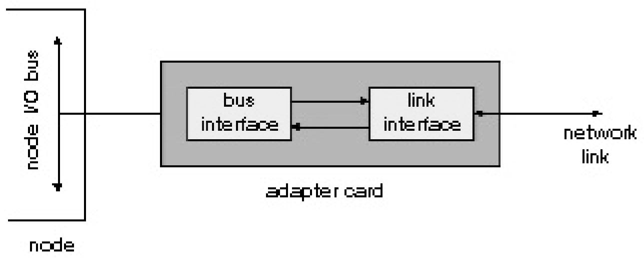


Figure 1-3: The adapter is a semi-autonomous unit.

As shown in Figure 1.3, the main components of an adapter are the bus interface and the link interface. The bus interface is responsible for communicating with the adapter's parent node. It sends to and receives from the parent node network-layer datagram's and control information. The link interface is responsible for implementing the link-layer protocol. In addition to framing and de-framing datagram's, it may provide error detection, random access and other link-layer functions. It also includes transmit and receive circuitry. For popular link-layer technologies, such as Ethernet, the link interface is implemented by chip set that can be bought on the commodity market. For this reason, Ethernet adapters are incredibly cheap often less than $30 for 10 Mbps and 100 Mbps transmission rates.

Adapter design has become very sophisticated over the years. One of the critical issues in adapter performance has always been whether the adapter can move data in and out of a node at the full line speed, that is, at the transmission rate of the link.

2. Error Detection and Correction Techniques

Structure

Objectives

- The types of errors that can be generated in frames during transmission;
- Error in a data frame;
- Methods for detecting errors;
- Methods for correcting errors;
- Forwarding the error correction method for error correction;
- Following the CRC method for error detection;

2-1 Introduction

In the previous lesson, we noted that bit-level error detection and correction detecting and correcting the corruption of bits in a data-link-layer frame sent from one node to another physically-connected neighboring node are two services often provided by the data link layer. In this section, we'll examine a few of the simplest techniques that can be used to detect and, in some cases, correct such bit errors. Our goal here is to develop an intuitive feel for the capabilities that error detection and correction techniques provide, and to see how a few simple techniques work and is used in practice in the data link layer.

Figure 2-1 illustrates the setting for our study. At the sending node, data, *D*, to be "protected" against bit errors is augmented with error detection and correction bits, *EDC*. Typically, the data to be protected includes not only the datagram passed down from the network layer for transmission across the link, but also link-level addressing information, sequence numbers, and other fields in the data link frame header. Both *D* and *EDC* are sent to the receiving node in a link-level frame. At the receiving node, a sequence of bits, *D*' and *EDC*' are received. Note that *D*' and *EDC*' may differ from the original *D* and *EDC* as a result of in-transit bit flips.

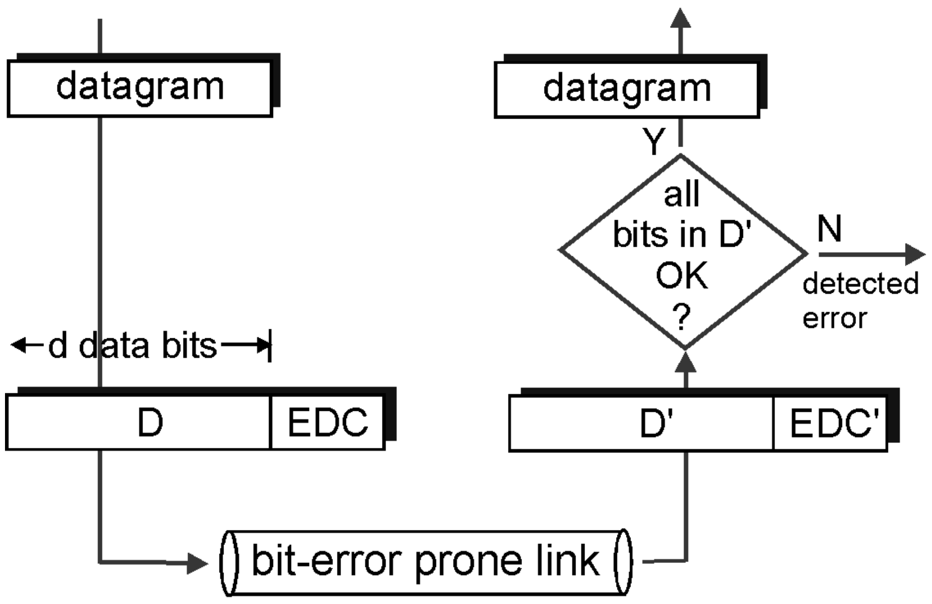


Figure 2-1: Error detection and correction scenario

The receiver's challenge is to determine whether or not *D*' is the same as the original *D*, given that it has only received *D*' and *EDC*'. The exact wording of the receiver's decision in Figure 2-1 (we ask whether an error is detected, not whether an error has occurred!) is important. Error detection and correction techniques allow the receiver to sometimes, *but not always*, detect that bit errors have occurred. That is, even with the use of error detection bits there will still be a possibility that **undetected bit errors** will occur, i.e., that the receiver will be unaware that the received information contains bit errors. As a consequence, the receiver might deliver a corrupted datagram to the network layer, or be unaware that the contents of some other field in the frame's header have been corrupted. We thus want to choose an error detection scheme so that the probability of such occurrences is small. Generally, more sophisticated error detection and correction techniques (i.e., those that have a smaller probability of allowing undetected bit errors) incur a larger overhead - more computation is need to compute and transmit a larger number of error detection and correction bits.

Let's now examine three techniques for detecting errors in the transmitted data parity checks (to illustrate the basic ideas behind error detection and correction), check summing methods (which are more typically employed in the transport layer) and cyclic redundancy checks (which are typically employed in the data link layer).

2-2 Parity Checks

Perhaps the simplest form of error detection is the use of a single **parity bit**. Suppose that the information to be sent, *D* in Figure 2-1, has *d* bits. In an even parity scheme, the sender simply includes one additional bit and chooses its value such that the total number of 1's in the *d+1* bits (the original information plus a parity bit) is even. For odd parity schemes, the parity bit value is chosen such that there are an odd number of 1's. Figure 2-2 illustrates an even parity scheme, with the single parity bit being stored in a separate field.

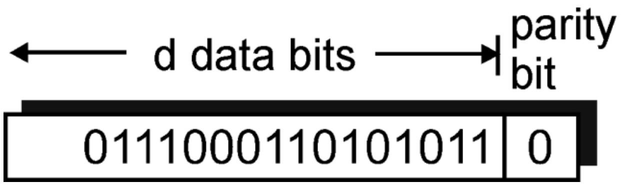
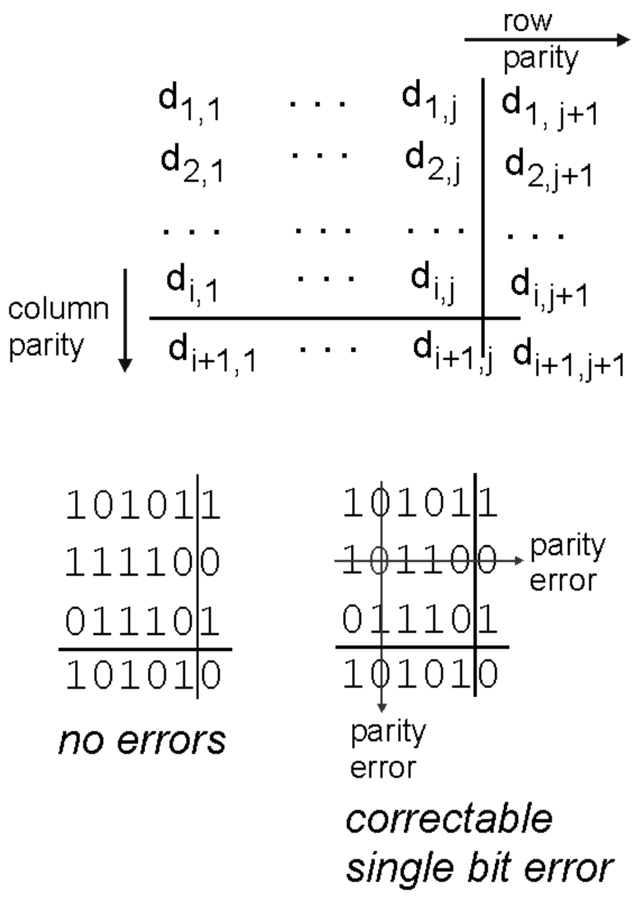


Figure 2-2: One-bit even parity

Receiver operation is also simple with a single parity bit. The receiver need only count the number of 1's in the received *d+ 1 bit*. If an odd number of 1-valued bits are found with an even parity scheme, the receiver knows that at least one bit error has occurred. More precisely, it knows that some *odd* number of bit errors has occurred.

But what happens if an even number of bit errors occurs? You should convince yourself that this would result in an undetected error. If the probability of bit errors is small and errors can be assumed to occur independently from one bit to the next, the probability of multiple bit errors in a packet would be extremely small. In this case, a single parity bit might suffice. However, measurements have shown that rather than occurring independently, errors are often clustered together in ``bursts.'' Under burst error conditions, the probability of undetected errors in a frame protected by single-bit-parity can approach 50 percent. Clearly, a more robust error detection scheme is needed (and, fortunately, is used in practice!). But before examining error detection schemes that are used in practice, let's consider a simple generalization of one-bit parity that will provide us with insight into error correction techniques.



row parity
column parity
$d_{1,1}$ . . . $d_{1,j}$ $d_{1,j+1}$
$d_{2,1}$ . . . $d_{2,j}$ $d_{2,j+1}$
$d_{i,1}$ . . . $d_{i,j}$ $d_{i,j+1}$
$d_{i+1,1}$ . . . $d_{i+1,j}$ $d_{i+1,j+1}$
10101 1
11110 0
01110 1
10101 0
no errors
10101 1
10110 0
01110 1
10101 0
parity error
parity error
correctable single bit error

Figure 2-3: Two-dimensional even parity

Figure 2-3 shows a two-dimensional generalization of the single-bit parity scheme. Here, the *d* bits in *D* are divided into *i* rows and *j* columns. A parity value is computed for each row and for each column. The resulting *i+j+1* parity bits are the data link frame's error detection bits.

Suppose now that a single bit error occurs in the original *d* bits of information. With this **two-dimensional parity** scheme, the parity of both the column and the row containing the flipped bit will be in error. The receiver can thus not only *detect* the fact that a single bit error has occurred, but can use the column and row indices of the column and row with parity errors to actually identify the bit that was corrupted and *correct* that error! Figure 2-3 shows an example in which the 0-valued bit in position (1, 1) is corrupted and switched to a 1 an error that is both detectable and correctable at the receiver. Although our discussion has focused on the original *d* bits of information, a single error in the parity bits themselves is also detectable and correctable. Two dimensional parity can also detect (but not correct!) any combination of two errors in a packet. Other properties of the two-dimensional parity scheme are explored in the problems at the end of the unit.

The ability of the receiver to both detect and correct errors is known as **forward error correction (FEC).** These techniques are commonly used in audio storage and playback devices such as audio CD's. In a network setting, FEC techniques can be used by themselves, or in conjunction with the ARQ techniques we examined in unit-II. FEC techniques are valuable because they can decrease the number of sender retransmissions required. Perhaps more importantly, they allow for immediate correction of errors at the receiver. This avoids having to wait the round-trip propagation delay needed for the sender to receive a NAK packet and for the retransmitted packet to propagate back to the receiver a potentially important advantage for real-time network applications.

2-3 Checksumming Methods

In checksumming techniques, the *d* bits of data in Figure 2-1 are treated as a sequence of *k*-bit integers. One simple checksumming method is to simply sum these *k*-bit integers and use the resulting sum as the error detection bits. The so-called **Internet checksum** is based on this approach bytes of data are treated as 16-bit integers and their ones-complement sum forms the Internet checksum. A receiver calculates the checksum it calculates over the received data and checks whether it matches the checksum carried in the received packet. In the TCP/IP protocols, the Internet checksum is computed over all fields (header and data fields included). In other protocols, e.g., XTP, one checksum is computed over the header, with another checksum computed over the entire packet.

2-4 cyclic redundancy check

An error detection technique used widely in today's computer networks is based on **cyclic redundancy check (CRC) codes**. CRC codes are also known as **polynomial codes**, since it is possible to view the bit string to be sent as a polynomial whose coefficients are the 0 and 1 values in the bit string, with operations on the bit string interpreted as polynomial arithmetic.

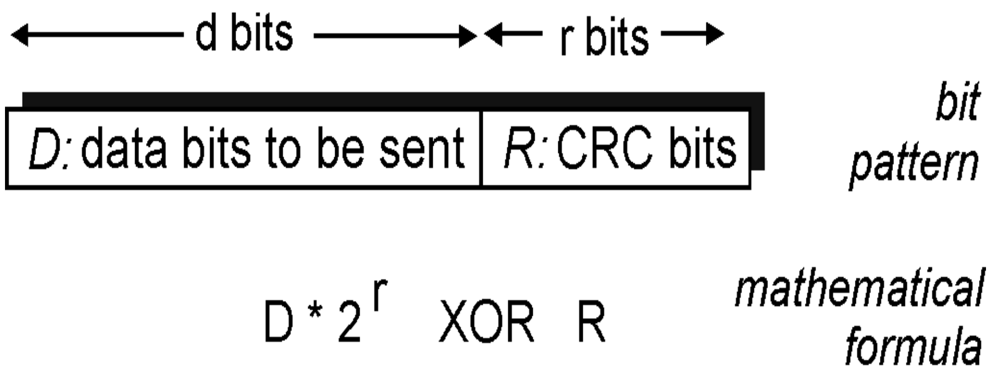


Figure 2-4: CRC codes

CRC codes operate as follows. Consider the *d-bit* piece of data, *D*, that the sending node wants to send to the receiving node. The sender and receiver must first agree on an *r+1* bit pattern, known as a **generator**, which we will denote as G. We will require that the most significant (leftmost) bit of *G* be a 1. The key idea behind CRC codes is shown in Figure 5.2-4. For a given piece of data, *D*, the sender will choose *r* additional bits, *R*, and append them to *D* such that the resulting *d+r* bit pattern (interpreted as a binary number) is exactly divisible by *G* using modulo-2 arithmetic. The process of error checking with CRC's is thus simple: the receiver divides the *d+r* received bits by *G.* If the remainder is non-zero, the receiver knows that an error has occurred; otherwise the data is accepted as being correct.

All CRC calculations are done in modulo-2 arithmetic without carries in addition or borrow in subtraction. This means that addition and subtraction are identical, and both are equivalent to the bitwise exclusive-or (XOR) of the operands. Thus, for example,

```
1011    XOR   0101   =       1110
1001    XOR   1101   =       0100
```

Also, we similarly have

```
1011    -     0101   =       1110
1001    -     1101   =       0100
```

Multiplication and division are the same as in base 2 arithmetic, except that any required addition or subtraction is done without carries or borrows. As in regular binary arithmetic, multiplication by 2^k left shifts a bit pattern by *k* places. Thus, given *D* and *R*, the quantity *D*2r XOR R* yields the *d+r* bit pattern shown in Figure 2-4. We'll use this algebraic characterization of the *d+r* bit pattern from Figure 2-4 in our discussion below.

Let us now turn to the crucial question of how the sender computes *R*. Recall that we want to find *R* such that there is an *n* such that

$$D*2^r \text{ XOR } R = nG$$

That is, we want to choose *R* such that *G* divides into *D*2r*XOR *R* without remainder. If we exclusive-or (i.e., add modulo 2, without carry) *R* to both sides of the above equation, we get

$$D*2^r = nG \text{ XOR } R$$

This equation tells us that if we divide $D*2^r$ by G, the value of the remainder is precisely *R*. In other words, we can calculate *R* as

$$R = \text{remainder } (D*2^r / G)$$

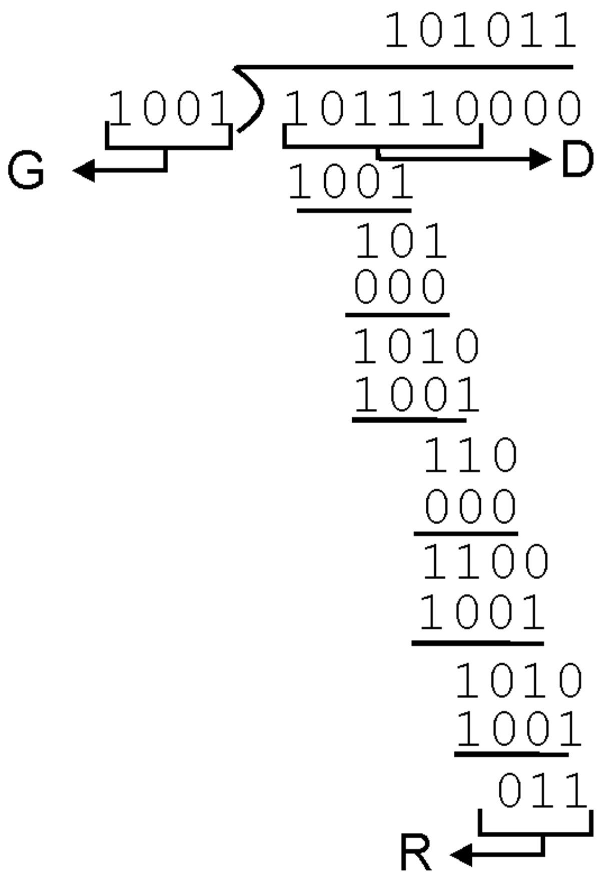


Figure 2-5: An example CRC calculation

Figure 2-5 illustrates this calculation for the case of D = 101110, *d* = 6 and G = 1001, r=3. The nine bits transmitted in this case are 101110 011. You should check these calculations for yourself and also check that indeed $D2^r$ = 101011 * *G* XOR *R*.

International standards have been defined for 8-, 12-, 16- and 32-bit generators, *G*. An 8-bit CRC is used to protect the 5-byte header in ATM cells. The CRC-32 32-bit standard, which has been adopted in a number of link-level IEEE protocols, uses a generator of

$$G_{CRC\text{-}32} = 100000100110000010001110110110111$$

Each of the CRC standards can detect burst errors of less than *r+1* bit and any odd number of bit errors. Furthermore, under appropriate assumptions, a burst of length greater than *r+1* bit is detected with probability $1 - 0.5^r$.

3. Multiple Access Protocols and LANs

Structure

Objectives

- The need for accessing multi-access channel;
- Common methods for accessing multi-access channel like FDM, TDM;
- The need for Dynamic channel allocation method;
- Pure ALOHA method for channel allocation;
- Slotted ALOHA method for channel allocation;
- Carrier sensing method CSMA to improve performance;
- Carrier sensing with collision detection method CSMA/CD;
- IEEE 802.3 standard and their Different Cabling types, and

.

3-1Introduction

In this lesson we'll take step back from specific link layer protocols and first examine a problem of central importance to the data link layer: how to coordinate the access of multiple sending and receiving nodes to a shared broadcast channel the so-called **multiple access problem**. Broadcast channels are often used in **local area networks (LANs)**, networks that are geographically concentrated in a single building (or on a corporate or university campus). Thus, we'll also look at how multiple access channels are used in LANs at the end of this lesson.

3-2 Multiple Access Channels (MAC)

We are all familiar with the notion of broadcasting, as television has been using it since its invention. But traditional television is a one-way broadcast (i.e., one fixed node transmitting to many receiving nodes), while nodes on a computer network broadcast channel can both send and receive. Perhaps a more apt human analogy for a broadcast channel is a cocktail party, where many people gather together in a large room (the air providing the broadcast medium) to talk and listen. A second good analogy is something many readers will be familiar with - a classroom where teacher(s) and student(s) similarly share the same, single, broadcast medium. A central problem in both scenarios is that of determining who gets to talk (i.e., transmit into the channel), and when. As humans, we've evolved an elaborate set of protocols for sharing the broadcast channel ("Give everyone a chance to speak." "Don't speak until you are spoken to." "Don't monopolize the conversation." "Raise your hand if you have question." "Don't interrupt when someone is speaking." "Don't fall asleep when someone else is talking.").

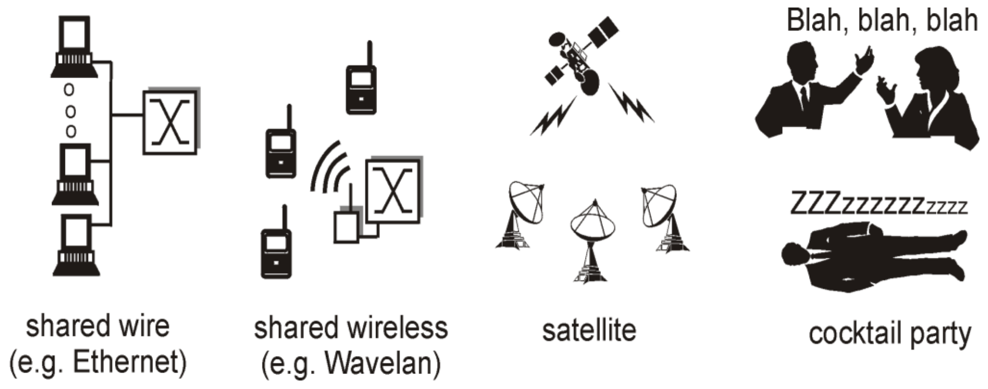


Figure 3-1: Various multiple access channels

Computer networks similarly have protocols - so-called multiple access protocols by which nodes regulate their transmission onto the shared broadcast channel. As shown in Figure 3-1, multiple access protocols are needed in a wide variety of network settings, including both wired and wireless local area networks, and satellite networks. Figure 3-2 takes a more abstract view of the broadcast channel and of the nodes sharing that channel. Although technically each node accesses the broadcast channel through its adapter, in this section we will refer to the *node* as the sending and receiving device. In practice, hundreds or even thousands of nodes can directly communicate over a broadcast channel.

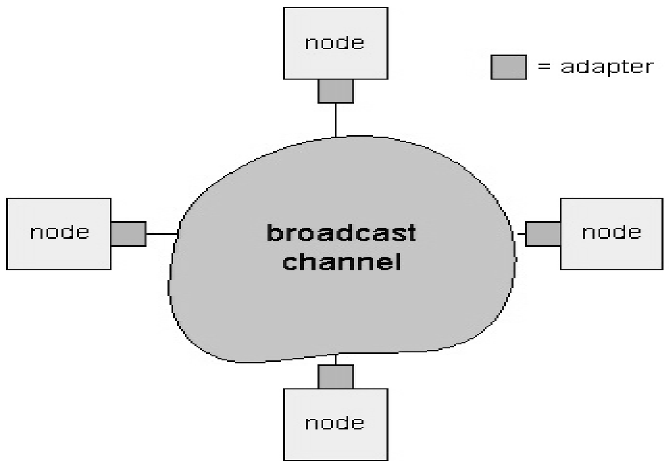


Figure 3-2: A broadcast channel interconnecting four nodes.

Because all nodes are capable of transmitting frames, more than two nodes can transmit frames at the same time. When this happens, all of the nodes receive multiple frames at the same time, that is, the transmitted frames **collide** at all of the receivers. Typically, when there is a collision, none of the receiving nodes can make any sense of any of the frames that were transmitted; in a sense, the signals of the colliding frame become inextricably tangled together. Thus, all the frames involved in the collision are lost, and the broadcast channel is wasted during the collision interval. Clearly, if many nodes want to frequently transmit frames, many transmissions will result in collisions, and much of the bandwidth of the broadcast channel will be wasted.

In order to ensure that the broadcast channel performs useful work when multiple nodes are active, it is necessary to somehow coordinate the transmissions of the active nodes. This coordination job is the responsibility of the **multiple access protocol**. Over the past thirty years, thousands of papers and hundreds of Ph.D. dissertations have been written on multiple access protocols; a comprehensive survey of this body of work is Furthermore, dozens of different protocols have been implemented in a variety of link-layer technologies. Nevertheless, we can classify just about any multiple access protocol as belonging to one of three categories: **channel partitioning protocols**, **random access protocols,** and **taking-turns protocols**. We'll cover these categories of multiple access protocols in the following three subsections. Let us conclude this overview by noting that ideally, a multiple access protocol for a broadcast channel of rate *R* bits per second should have the following desirable characteristics:

1. When only one node has data to send, that node has a throughput of *R* bps.
2. When *M* nodes have data to send, each of these nodes has a throughput of *R/M* bps. This need not necessarily imply that each of the *M* nodes always have an instantaneous rate of *R/M* , but rather that each node should have an average transmission rate of *R/M* over some suitably-defined interval of time.
3. The protocol is decentralized, i.e., there are no master nodes that can fail and bring down the entire system.
4. The protocol is simple, so that it is inexpensive to implement.

3-3 Channel Partitioning Protocols

Recall from our early discussion back in unit-I, that Time Division Multiplexing (TDM) and Frequency Division Multiplexing (FDM) are two techniques that can be used to partition a broadcast channel's bandwidth among all nodes sharing that channel. As an example, suppose the channel supports *N* nodes and that the transmission rate of the channel is *R* bps. TDM divides time into **time frames** (not to be confused the unit of data, the frame, at the data link layer) and further divides each time frame into *N* **time slots**. Each slot time is then assigned to one of the *N* nodes. Whenever a node has a frame to send, it transmits the frame's bits during its assigned time slot in the revolving TDM frame. Typically, frame sizes are chosen so that a single frame can be transmitting during a slot time. Figure 3-3 shows a simple four-node TDM example. Returning to our cocktail party analogy, a TDM-regulated cocktail party would allow one partygoer to speak for a fixed period of time, and then allow another partygoer to speak for the same amount of time, and so on. Once everyone has had their chance to talk, the pattern repeats.

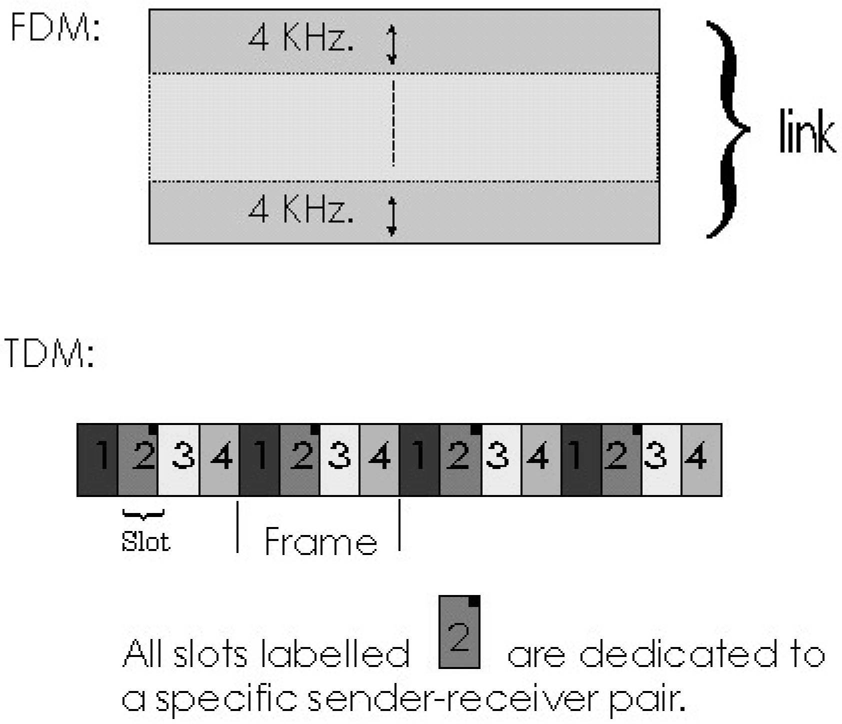

FDM:
4 KHz.
4 KHz.
link
TDM:
1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
Slot
Frame
All slots labelled 2 are dedicated to a specific sender-receiver pair.

Figure 3-3: A four-node TDM and FDM example

TDM is appealing as it eliminates collisions and is perfectly fair: each node gets a dedicated transmission rate of *R/N* bps during each slot time. However, it has two major drawbacks. First, a node is limited to this rate of *R/N* bps over a slot's time even when it is the only node with frames to send. A second drawback is that a node must always wait for its turn in the transmission sequence again, even when it is the only node with a frame to send. Imagine the partygoer who is the only one with anything to say (and imagine that this is the even rarer circumstance where everyone at the party wants to hear what that one person has to say). Clearly, TDM would be a poor choice for a multiple access protocol for this particular party.

While TDM shares the broadcast channel in time, FDM divides the *R* bps channel into different frequencies (each with a bandwidth of *R/N*) and assigns each frequency to one of the *N* nodes. FDM thus creates *N* "smaller" channels of *R/N* bps out of the single, "larger" Rbps channel.

FDM shares both the advantages and drawbacks of TDM. It avoids collisions and divides the bandwidth fairly among the *N* nodes. However, FDM also shares a principal disadvantage with TDM - a node is limited to a bandwidth of R/N, even when it is the only node with frames to send.

A third channel partitioning protocol is **Code Division Multiple Access (CDMA).** While TDM and FDM assign times slots and frequencies, respectively, to the nodes, CDMA assigns a different *code* to each node. Each node then uses its unique code to encode the data bits it sends, as discussed below. We'll see that CDMA allows different nodes to transmit *simultaneously* and yet have their respective receivers correctly receive a sender's encoded data bits (assuming the receiver knows the sender's code) in spite of "interfering" transmissions by other nodes. CDMA has been used in military systems for some time (due its anti jamming properties) and is now beginning to find widespread civilian use, particularly for uses in wireless multiple access channels.

In a CDMA protocol, each bit being sent by the sender is encoded by multiplying the bit by a signal (the code) that changes at a much faster rate (known as the **chipping rate**) than the original sequence of data bits. Figure 3-4 shows a simple, idealized CDMA encoding/decoding scenario. Suppose that the rate at which original data bits reach the CDMA encoder defines the unit of time; that is, each original data bit to be transmitted requires one bit-slot time. Let d_i be the value of the data bit for the *i*th bit slot. Each bit slot is further subdivided into *M* mini-slots; in Figure 5.3-4, *M=8*, although in practice *M* is much larger. The CDMA code used by the sender consists of a sequence of M values, c_m, $m = 1... M$, each taking a +1 or -1 value. In the example in Figure 5.3-4, the *M*-bit CDMA code being used by the sender is (1, 1, 1, -1, 1, -1, -1, -1).

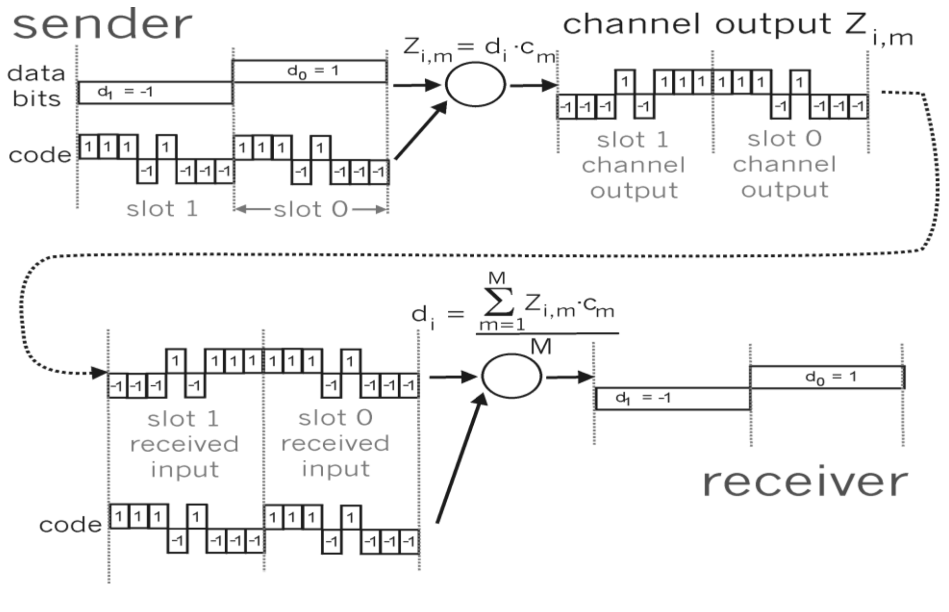


Figure 3-4: A simple CDMA example: sender encoding, receiver decoding

To illustrate how CDMA works, let us focus on the *i*th data bit, d_i. For the *m*th mini-slot of the bit-transmission time of d_i, the output of the CDMA encoder, $Z_{i,m}$, is the value of d_i multiplied by the *m*th bit in the assigned CDMA code, c_m:

$$Z_{i,m} = d_i \cdot c_m \qquad \text{(Equation 3-1)}$$

In a simple world, with no interfering senders, the receiver would receive the encoded bits, $Z_{i,m}$, and recover the original data bit, d_i, by computing:

$$d_i = (1/M) \sum_{m=1}^{M} Zi,m \quad c_m \qquad \text{(Equation 3-2)}$$

The reader might want to work through the details of the example in Figure 3-4 to see that the original data bits are indeed correctly recovered at the receiver using Equation 3-2

The world is far from ideal, however, and as noted above, CDMA must work in the presence of interfering senders that are encoding and transmitting their data using a different assigned code. But how can a CDMA receiver recover a sender's original data bits when those data bits are being tangled with bits being transmitted by other senders? CDMA works under the assumption that the interfering transmitted bit signals are additive, e.g., that if three senders send a 1 value, and a fourth sender sends a -1 value during the same mini-slot, then the received signal at all receivers during hat mini-slot is a 2 (since 1+ 1 + 1 - 1 = 2). In the presence of multiple senders, sender *s* computes its encoded transmissions, $Z_{i,m}{}^{s}$, in exactly the same manner as in Equation 5.3-1. The value received at a receiver during the *m*th minis lot of the *i*th bit slot, however, is now the *sum* of the transmitted bits from all *N* senders during that minis lot:

$$Z_{i,m}{}^{*} = \sum_{s=1}^{N} \mathbf{Z}^{s}_{\mathbf{i,m}}$$

Amazingly, if the senders' codes are chosen carefully, each receiver can recover the data sent by a given sender out of the aggregate signal simply by using the sender's code in exactly the same manner as in Equation 5.3-2:

$$d_i = (1/M) \sum_{m=1}^{M} \quad Z_{im}{}^{*} \cdot c_m \qquad \text{(Equation 3-3)}$$

Figure 3-5 illustrates a two-sender CDMA example. The *M*-bit CDMA code being used by the upper sender is (1, 1, 1, -1, 1, -1, -1, -1), while the CDMA code being used by the lower sender is (1, -1, 1, 1, 1, -1, 1, 1). Figure 3-5 illustrates a receiver recovering the original data bits from the upper sender. Note that the receiver is able to extract the data from sender 1 in spite of the interfering transmission from sender 2. Returning to our cocktail party analogy, a CDMA protocol is similar to having partygoers speaking in multiple languages; in such circumstances humans are actually quite good at locking into the conversation in the language they understand, while filtering out the remaining conversations.

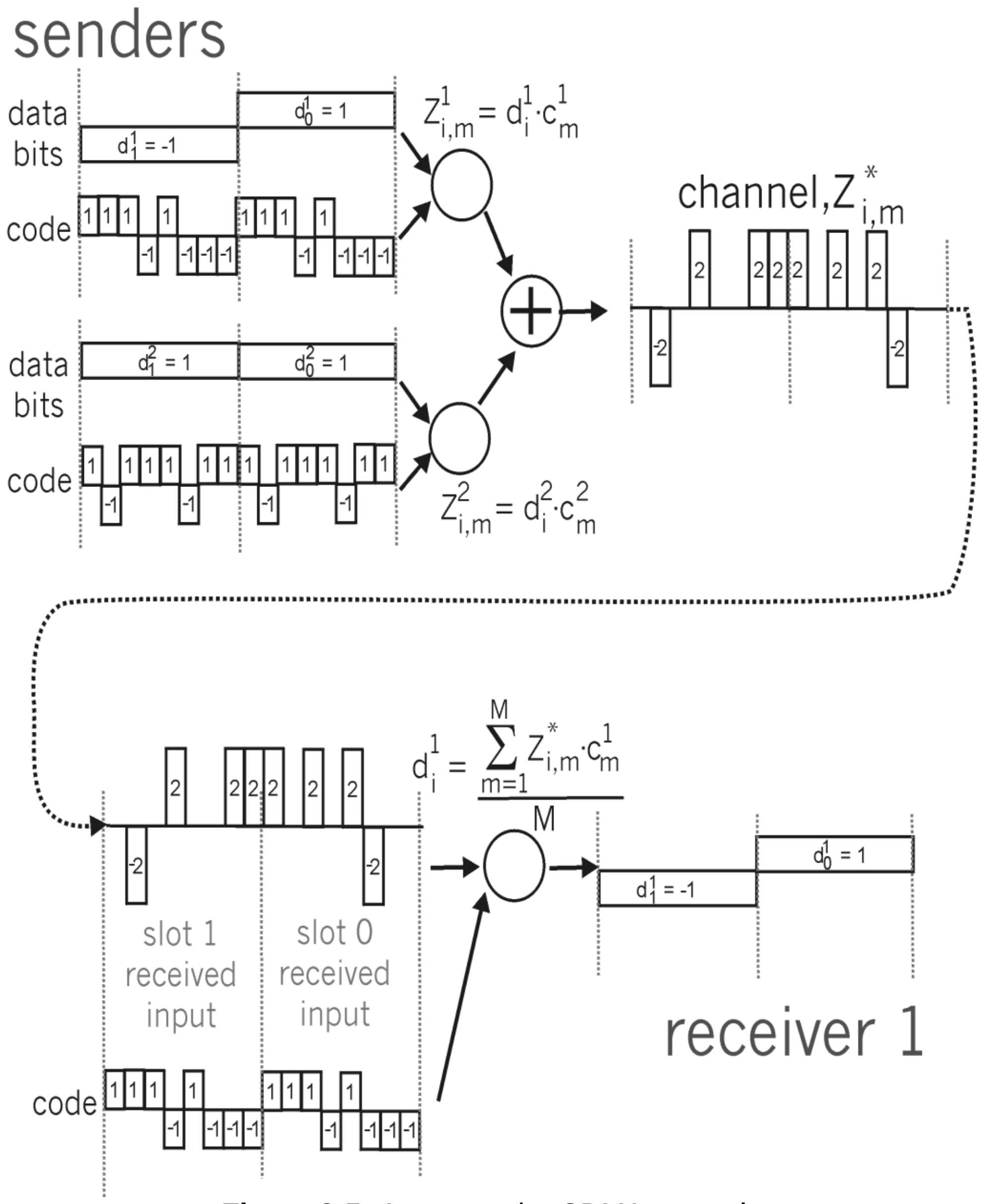


Figure 3-5: A two-sender CDMA example

Our discussion here of CDMA is necessarily brief and a number of difficult issues must be addressed in practice. First, in order for the CDMA receivers to be able to extract out a particular sender's signal, the CDMA codes must be carefully chosen. Secondly, our discussion has assumed that the received signal strengths from various senders at a receiver are the same; this can be difficult to achieve in practice. There is a considerable body of literature addressing these and other issues related to CDMA.

3-4 Random Access Protocols

The second broad class of multiple access protocols is so-called random access protocols. In a random access protocol, a transmitting node always transmits at the full rate of the channel, namely, R bps. When there is a collision, each node involved in the collision repeatedly retransmits its frame until the frame gets through without a collision. But when a node experiences a collision, it doesn't necessarily retransmit the frame right away. *Instead it waits a random delay before retransmitting the frame*. Each node involved in a collision chooses independent random delays. Because after a collision the random delays are independently chosen, it is possible that one of the nodes will pick a delay that is sufficiently less than the delays of the other colliding nodes, and will therefore be able to "sneak" its frame into the channel without a collision.

3-4-1Slotted ALOHA

Let's begin our study of random access protocols with one of the most simple random access protocols, the so-called slotted ALOHA protocol. In our description of slotted ALOHA, we assume the following:

- All frames consist of exactly *L* bits.
- Time is divided into slots of size *L/R* seconds (i.e., a slot equals the time to transmit one frame).
- Nodes start to transmit frames only at the beginnings of slots.
- The nodes are synchronized so that each node knows when the slots begin.
- If two or more frames collide in a slot, then all the nodes detect the collision event before the slot ends.

Let *p* be a probability, that is, a number between 0 and 1. The operation of slotted ALOHA in each node is simple:

- When the node has a fresh frame to send, it waits until the beginning of the next slot and transmits the entire frame in the slot.
- If there isn't a collision, the node won't consider retransmitting the frame. (The node can prepare a new frame for transmission, if it has one.)
- If there is a collision, the node detects the collision before the end of the slot. The node retransmits its frame in each subsequent slot with probability *p* until the frame is transmitted without a collision.

By retransmitting with probability *p*, we mean that the node effectively tosses a biased coin; the event heads corresponds to retransmit, which occurs with probability *p.* The event tails corresponds to "skip the slot and toss the coin again in the next slot"; this occurs with probability *(1-p).* Each of the nodes involved in the collision toss their coins independently.

Slotted ALOHA would appear to have many advantages. Unlike channel partitioning, slotted ALOHA allows a single active node (i.e., a node with a frame to send) to continuously transmit frames at the full rate of the channel. Slotted ALOHA is also highly decentralized, as each node detects collisions and independently decides when to retransmit. (Slotted ALOHA does, however, require the slots to be synchronized in the nodes; we'll shortly discuss an unslotted version of the ALOHA protocol, as well as CSMA protocols; one of which require such synchronization and are therefore fully decentralized.) Slotted ALOHA is also an extremely simple protocol.

Slotted ALOHA also works great when there is only one active node, but how efficient is it when there are multiple active nodes? There are two possible efficiency concerns here. First, as shown in Figure 3-6, when there are multiple active nodes, a certain fraction of the slots will have *collisions* and will therefore be "wasted." The second concern is that another fraction of the slots will be *empty* because all active nodes refrain from transmitting as a result of the probabilistic transmission policy. The only "un-wasted" slots will be those in which exactly one node transmits. A slot in which exactly one node transmits is said to be a **successful slot**. The **efficiency** of a slotted multiple access protocol is defined to be the long-run fraction of successful slots when there are a large number of active nodes, with each node having a large number of frames to send. Note that if no form of access control were used, and each node were to immediately retransmits after each collision, the efficiency would be zero. Slotted ALOHA clearly increases the efficiency beyond zero, but by how much?

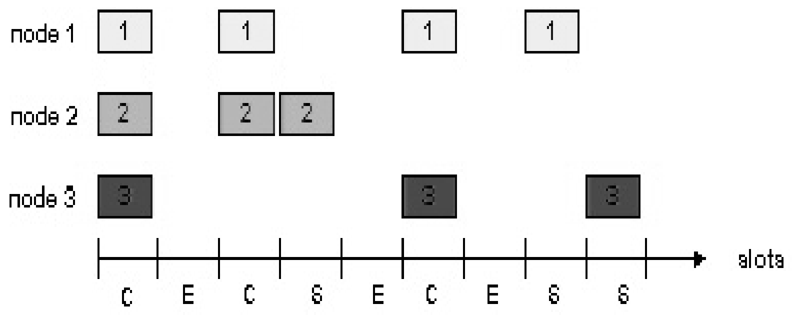


Figure 5.3-6: Nodes 1, 2 and 3 collide in the first slot. Node 2 finally succeeds in the fourth slot, node 1 in the eighth slot, and node 3 in the ninth slot. The notation C, E and S represent "collision slot", "empty slot" and "successful slot", respectively

We now proceed to outline the derivation of the maximum efficiency of slotted ALOHA. To keep this derivation simple, let's modify the protocol a little and assume that each node attempts to transmit a frame in each slot with probability p. (That is, we assume that each node always has a frame to send and that the node transmits with probability *p* for a fresh frame as well as for a frame that has already suffered a collision.) Suppose first there are *N* nodes. Then the probability that a given slot is a successful slot is the probability that one of the nodes transmits and that the remaining *N-1* nodes do not transmit. The probability that a given node transmits is *p*; the probability that the remaining nodes do not transmit is $(1-p)^{N-1}$. Therefore the probability a given node has a success is $p\,(1-p)^{N-1}$. Because there are *N* nodes, the probability that an arbitrary node has a success is $Np(1-p)^{N-1}$.

Thus, when there are N active nodes, the efficiency of slotted ALOHA is $Np(1-p)^{N-1}$. To obtain the *maximum* efficiency for *N* active nodes, we have to find the *p** that maximizes this expression. (See the homework problems for a general outline of this derivation.) And to obtain the maximum efficiency for a large number of active nodes, we take the limit of $Np^*(1-p^*)^{N-1}$ as *N* approaches infinity. (Again, see homework problems.) After performing these calculations, we'll find that the maximum efficiency of the protocol is given by $1/e = .37$. That is, when a large number of nodes have many frames to transmit, then (at best) only 37% of the slots do useful work. Thus the effective transmission rate of the channel is not R bps but only .37 R bps! A similar analysis also shows that 37% of the slots go empty and 26% of slots have collisions. Imagine the poor network administrator who has purchased a 100 Mbps slotted ALOHA system, expecting to be able to use the network to transmit data among a large number of users at an aggregate rate of, say, 80 Mbps! Although the channel is capable of transmitting a given frame at the full channel rate of 100Mbps, in the long term, the successful throughput of this channel will be less that 37 Mbps.

3-4-2 ALOHA

The slotted ALOHA protocol required that all nodes synchronize their transmissions to start at the beginning of a slot. The first ALOHA protocol [Abramson 1970] was actually an unslotted, fully decentralized, protocol. In so-called pure ALOHA, when a frame first arrives (i.e., a network layer datagram is passed down from the network layer at the sending node), the node immediately transmits the frame in it's entirely into the broadcast channel. If a transmitted frame experiences a collision with one or more other transmissions, the node will then immediately (after completely transmitting its collided frame) retransmit the frame with probability *p.* Otherwise, the node waits for a frame transmission time. After this wait, it then transmits the frame with probability *p*, or waits (remaining idle) for another frame time with probability *1-p*.

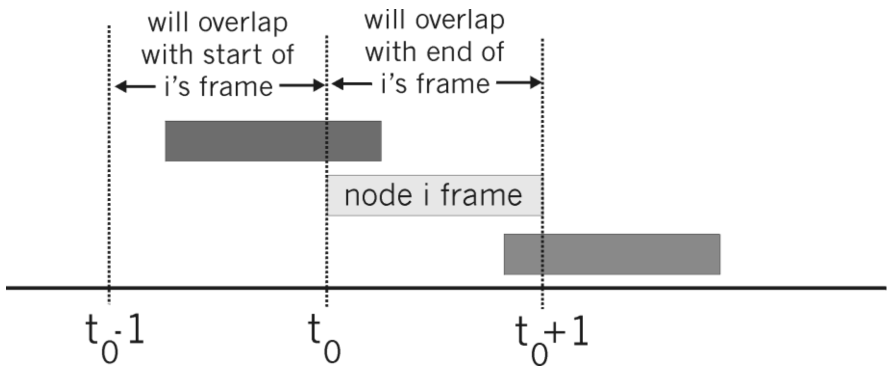

will overlap with start of i's frame
will overlap with end of i's frame
node i frame
t_0-1
t_0
t_0+1

Figure 3-7: Interfering transmissions in pure Aloha

To determine the maximum efficiency of pure ALOHA, we focus on an individual node. We'll make the same assumptions as in our slotted ALOHA analysis and take the frame transmission time to be the unit of time at any given time, the probability that a node is transmitting a frame is *p.* suppose this frame begins transmission at time t_0. As shown in Figure 5.3-7, in order for this frame to be successfully transmitted, no other nodes can begin their transmission in the interval of time [t_0-1, t_0]. Such a transmission would overlap with the beginning of the transmission of node *i*'s frame. The probability that all other nodes do not begin a transmission in this interval is $(1-p)^{N-1}$. Similarly, no other node can begin a transmission while node *i* is transmitting, as such a transmission would overlap with the latter part of node *i*'s transmission. The probability that all other nodes do not begin a transmission in this interval is also $(1-p)^{N-1}$. Thus, the probability that a given node has a successful transmission is $p(1-p)^{2(N-1)}$. By taking limits as in the slotted ALOHA case, we find that the maximum efficiency of the pure ALOHA protocol is only *1/ (2e)* - exactly half that of slotted ALOHA. This then is the price to be paid for a fully decentralized ALOHA protocol.

3-4-3 CSMA - Carrier Sense Multiple Access

In both slotted and pure ALOHA, a node's decision to transmit is made independently of the activity of the other nodes attached to the broadcast channel. In particular, a node neither pays attention to whether another node happens to be transmitting when it begins to transmit, nor stops transmitting if another node begins to interfere with its transmission. In our cocktail party analogy, ALOHA protocols are quite like a boorish partygoer who continues to chatter away regardless of whether other people are talking. As humans, we have human protocols that allow allows us to not only behave with more civility, but also to decrease the amount of time spent "colliding" with each other in conversation and consequently increasing the amount of amount of data we exchange in our conversations. Specifically, there are two important rules for polite human conversation:

- *Listen before speaking.* If someone else is speaking, wait until they are done. In the networking world, this is termed **carrier sensing** -

a node listens to the channel before transmitting. If a frame from another node is currently being transmitted into the channel, a node then waits ("backs off") a random amount of time and then again senses the channel. If the channel is sensed to be idle, the node then begins frame transmission. Otherwise, the node waits another random amount of time and repeats this process.

- *If someone else begins talking at the same time, stop talking.* In the networking world, this is termed **collision detection** - a transmitting node listens to the channel while it is transmitting. If it detects that another node is transmitting an interfering frame, it stops transmitting and uses some protocol to determine when it should next attempt to transmit.

These two rules are embodied in the family of CSMA (Carrier Sense Multiple Access) and **CSMA/CD** (CSMA with Collision Detection) protocols. Many variations on CSMA and CSMA/CD have been proposed, with the differences being primarily in the manner in which nodes perform backoff. The reader can consult these references for the details of these protocols. We'll study the CSMA/CD scheme used in Ethernet in detail in Section 5.5. Here, we'll consider a few of the most important, and fundamental, characteristics of CSMA and CSMA/CD.

The first question that one might ask about CSMA is that if all nodes perform carrier sensing, why do collisions occur in the first place? After all, a node will refrain from transmitting whenever it senses that another node is transmitting. The answer to the question can best be illustrated using space-time diagrams. Figure 3-7 shows a space-time diagram of four nodes (A, B, C, D) attached to an linear broadcast bus. The horizontal axis shows the position of each node in space; the y-axis represents time.

At time t_0, node B senses the channel is idle, as no other nodes are currently transmitting. Node B thus begins transmitting, with its bits propagating in both directions along the broadcast medium. The downward propagation of B's bits in Figure 3-7 with increasing time indicates that a non-zero amount of time is needed for B's bits to actually propagate (albeit at near the speed-of-light) along the broadcast medium. At time t_1 ($t_1 > t_0$), node D has a frame to send. Although node B is currently transmitting at time t_1, the bits being transmitted by B have yet to reach D, and thus D senses the channel idle at t_1. In accordance with the CSMA protocol, D thus begins transmitting its frame. A short time later, B's transmission begins to interfere with D's transmission at D. From Figure 3-7, it is evident that the end-to-end **channel propagation delay** of a broadcast channel - the time it takes for a signal to propagate from one of the channel to another will play a crucial role in determining its performance. Longer this propagation delay, the larger the chance that a carrier-sensing node is not yet able to sense a transmission that has already begun at another node in the network.

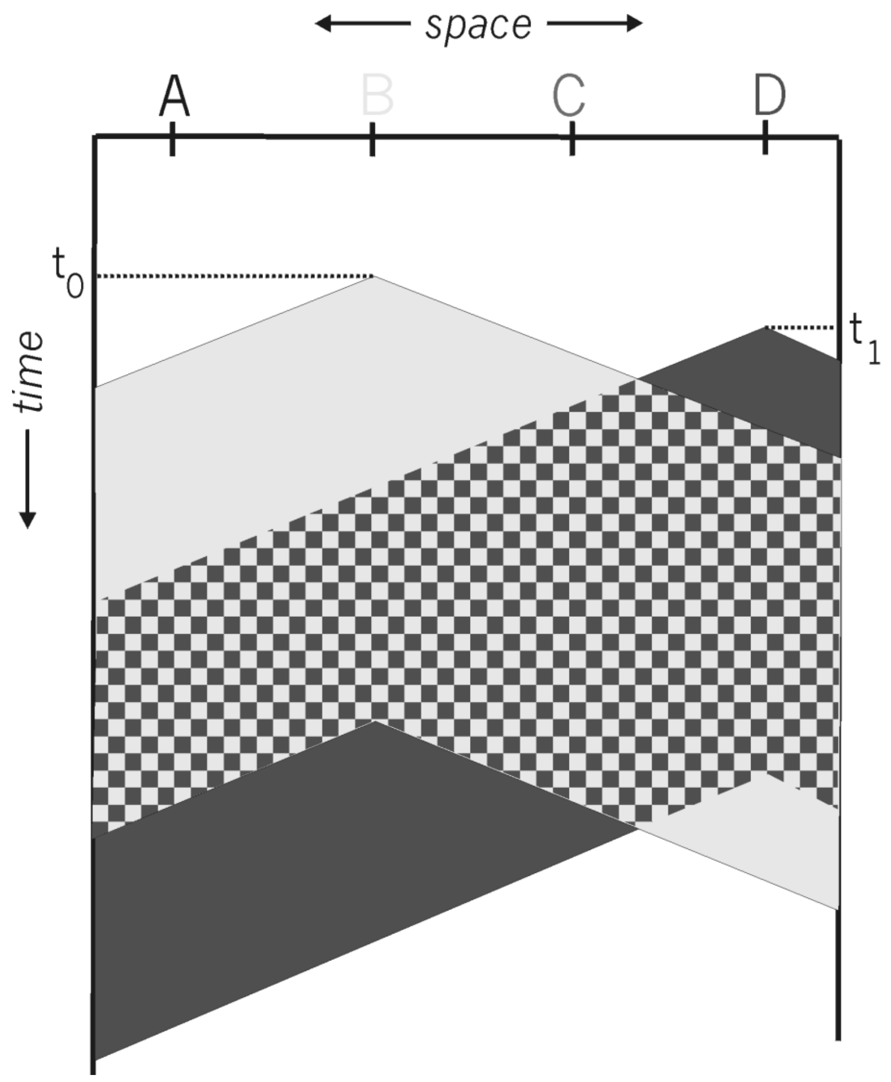


Figure 3-7: Space-time diagram of two CSMA nodes with colliding transmissions

In Figure 3-7, nodes do not perform collision detection; both B and D continue to transmit their frames in their entirety even though a collision has occurred. When a node performs collision detection it will cease transmission as soon as it detects a collision. Figure 3-8 shows the same scenario as in Figure 3-7, except that the two nodes each abort their transmission a short time after detecting a collision. Clearly, adding collision detection to a multiple access protocol will help protocol performance by not transmitting a useless, damaged (by interference with a frame from another node) frame in its entirety. The Ethernet protocol we will study in lesson-4 is a CSMA protocol that uses collision detection.

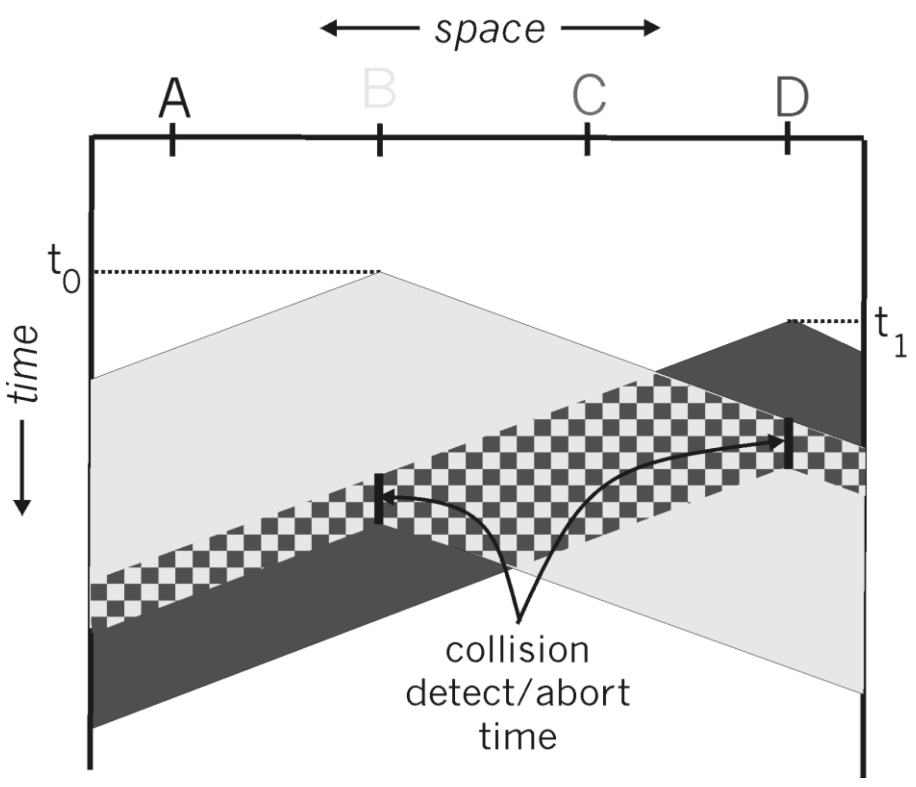


Figure 3-8: CSMA with collision detection.

3-5 Taking-Turns Protocols

Recall that two desirable properties of a multiple access protocol are *(i)* when only one node is active, the active node has a throughput of R bps, and *(ii)* when *M* nodes are active, then each active node has a throughput of nearly *R/M* bps. The ALOHA and CSMA protocols have this first property but not the second. This has motivated researchers to create another class of protocols the **taking-turns protocols**. As with random-access protocols, there are dozens of taking-turns protocols and each one of these protocols has many variations. We'll discuss two of the more important protocols here. The first one is the **polling protocol**. The polling protocol requires one of the nodes to be designated as a "master node" (or requires the introduction of a new node serving as the master). The master node **polls** each of the nodes in a round-robin fashion. In particular, the master node first sends a message to node 1, saying that it can transmit up to some maximum number of frames. After node 1 transmits some frames (from zero up to the maximum number), the master node tells node 2 it can transmit up to the maximum number of frames. (The master node can determine when a node has finished sending its frames by observing the lack of a signal on the channel.) The procedure continues in this manner, with the master node polling each of the nodes in a cyclic manner.

The polling protocol eliminates the collisions and the empty slots that plague the random access protocols. This allows it to have a much higher efficiency. But it also has a few drawbacks. The first drawback is that the protocol introduces a polling delay, the amount of time required to notify a node that it can transmit. If, for example, only one node is active, then the node will transmit at a rate less than R bps, as the master node must poll each of the inactive nodes in turn, each time the active node sends its maximum number of frames. The second drawback, which is potentially more serious, is that if the master node fails, the entire channel becomes inoperative.

The second taking-turn protocol is the **token-passing protocol**. In this protocol there is no master node. A small, special-purpose frame known as a **token** is exchanged among the nodes in some fixed order. For example, node 1 might always send the token to node 2, node 2 might always send the token to node 3, and node N might always send the token to node 1. When a node receives a token, it holds onto the token only if it has some frames to transmit; otherwise, it immediately forwards the token to the next node. If a node does have frames to transmit when it receives the token, it sends up to a maximum number of frames and then forwards the token to the next node. Token passing is decentralized and has a high efficiency. But it has its problems as well. For example, the failure of one node can crash the entire channel. Or if a node accidentally neglects to release the token, then some recovery procedure must be invoked to get the token back in circulation? Over the years many token-passing products have been developed, and each one had to address these as well as other sticky issues.

3-6 Local Area Networks

Multiple access protocols are used in conjunction with many different types of broadcast channels. They have been used for satellite and wireless channels, whose nodes transmit over a common frequency spectrum. They are currently used in the upstream channel for cable access to the Internet. And they are extensively used in local area networks (LANs).

Recall that a **LAN** is a computer network that is concentrated in a geographical area, such as in a building or on a university campus. When a user accesses the Internet from a university or corporate campus, the access is almost always by way of a LAN. For this type of Internet access, the user's host is a node on the LAN, and the LAN provides access to the Internet through a router, as shown in Figure 3-9. The LAN is a single "link" between each user host and the router; it therefore uses a link-layer protocol, which incorporates a multiple access protocol. The transmission rate, R, of most LANs is very high. Even in the early 1980s, 10 Mbps LANs were common; today, 100 Mbps LANs are common, and 1 Gbps LANs are available.

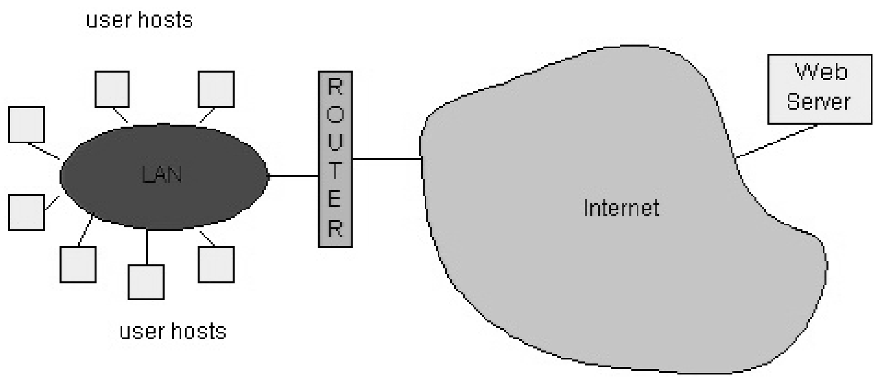


Figure 3-9: User hosts access an Internet Web server through a LAN. The broadcast channel between a user host and the router consists of one "link".

In the 1980s and the early 1990s, two classes of LAN technologies were popular in the workplace. The first class consists of the Ethernet LANs (also known as 802.3 LANs, which are random-access based. The second class of LAN technologies is token-passing technologies, including *token ring* (also known as IEEE 802.5 and *FDDI* (also known as Fiber Distributed Data Interface). Because we shall explore the Ethernet technologies in some detail in lesson-4, we focus our discussion here on the token-passing LANs. Our discussion on token-passing technologies is intentionally brief, since these technologies have become relatively minor players in the face of relentless Ethernet competition. Nevertheless, in order to provide examples about token-passing technology and to give a little historical perspective, it is useful to say a few words about token rings.

In a token ring LAN, the *N* nodes of the LAN (hosts and routers) are connected in a ring by direct links. The topology of the token ring defines the token-passing order. When a node obtains the token and sends a frame, the frame propagates around the entire ring, thereby creating a virtual broadcast channel. The node that sends the frame has the responsibility of removing the frame from the ring. FDDI was designed for geographically larger LANs (so called MANs, that is, metropolitan area networks). For geographically large LANs (spread out over several kilometers) it is inefficient to let a frame propagate back to the sending node once the frame has passed the destination node. FDDI has the destination node remove the frame from the ring. (Strictly speaking, FDDI is not a pure broadcast channel, as every node does not receive every transmitted frame.)

4. LAN Addresses and ARP

Structure

Objectives

- Understand LAN address;
- Describe the operation conversion of ARP

4-1 Introduction

As we learned in the previous lesson, nodes in LANs send frames to each other over a broadcast channel. This means that when a node in a LAN transmits a frame, every other node connected to the LAN receives the frame. But usually, a node in the LAN doesn't want to send a frame to *all of* the other LAN nodes but instead wants to send to one *particular* LAN node. To provide this functionality, the nodes on the LAN must be able to address each other when sending frames; that is, the nodes need LAN addresses and the link-layer frame needs a field to contain such a destination address. In this manner, when a node receives a frame, it can determine whether the frame was intended for it or for some other node in the LAN:

- If the destination address of the frame matches a receiving node's LAN address, then the node extracts the network-layer datagram from the link-layer frame and passes the datagram up the protocol stack.

- If the destination address does not match the address of the receiving node, the node simply discards the frame.

4-2 LAN Addresses

In truth, it is not a node that has a LAN address but instead a node's adapter that has a LAN address. This is illustrated in Figure 4.1. A LAN **address** is also variously called a **physical address,** an **Ethernet address,** or a **MAC** (Media Access Control) address. For most LANs (including Ethernet and token-passing LANs), the LAN address is six bytes long, giving 2^{48} possible LAN addresses. As shown in Figure 4.1, these six-byte addresses are typically expressed in hexadecimal notation, with each byte of the address expressed as a pair of hexadecimal numbers. An adapter's LAN address is permanent when an adapter is manufactured; a LAN address is burned into the adapter's ROM,

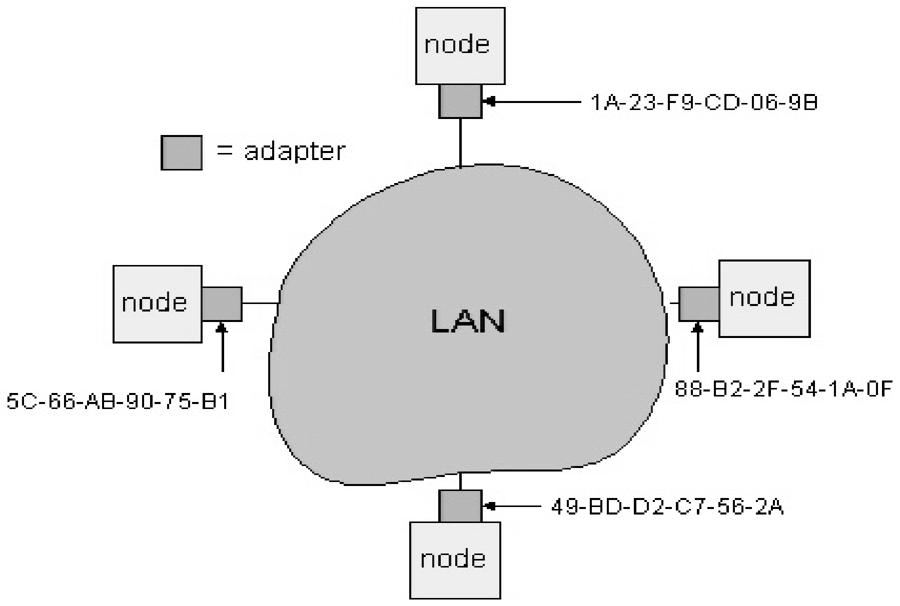


Figure 4-1: Each adapter connected to a LAN has a unique LAN address

One interesting property of LAN addresses is that no two adapters have the same address. This might seem surprising given that adapters are manufactured in many different countries by many different companies. How doe’s a company manufacturing adapters in Taiwan make sure that it is using different addresses from a company manufacturing adapters in Belgium? The answer is that IEEE manages the physical address space. In particular, when a company wants to manufacture adapters, it purchases a chunk of the address space consisting of 2^{24} addresses for a nominal fee. IEEE allocates the chunk of 2^{24} addresses by fixing the first 24 bits of a physical address and letting the company create unique combinations of the last 24 bits for each adapter.

An adapter's LAN address has a flat structure (as opposed to a hierarchical structure), and doesn't change no matter where the adapter goes. A portable computer with an Ethernet card always has the same LAN address, no matter where the computer goes. Recall that, in contrast, an IP address has a hierarchical structure (that is, a network part and a host part), and a node's IP address needs to be changed when the host moves. An adapter's LAN address is analogous to a person's social security number, which also has a flat addressing structure and which doesn't change no matter where the person goes. An IP address is analogous to a person's postal address, which is hierarchical and which needs to be changed whenever a person moves.

As we described at the beginning of this section, when an adapter wants to send a frame to some destination adapter on the same LAN, the sending adapter inserts the destination's LAN address into the frame. When the destination adapter receives the frame, it extracts the enclosed datagram and passes the datagram up the protocol stack. All the other adapters on the LAN also receive the frame. However, these other adapters discard the frame without passing the network-layer datagram up the protocol stack. Thus, these other adapters do not have to interrupt their parent node when they receive datagram's destined to other nodes. However, sometimes a sending adapter *does* want all the other adapters on the LAN to receive and *process* the frame it is about to send. In this case, the sending adapter inserts a special LAN **broadcast address** into the destination address field of the frame. For; LANs that use six-byte addresses (such as Ethernet and token-passing LANs), the broadcast address is a string of 48 consecutive is (that is, FF-FF-FF-FF-FF-FF in' hexadecimal notation).

4-3 Address Resolution Protocol

Because there are both network-layer addresses (for example, Internet IP addresses) and link-layer addresses (that is, LAN addresses), there is a need to translate between them. For the Internet, this is the job of the **address resolution protocol (ARP)** [RFC 826]. Every Internet host and router on a LAN has an **ARP module.**

To motivate ARP, consider the network shown in Figure 4-2. In this simple example, each node has a single IP address, and each node's adapter has a LAN address. As usual, IP addresses are shown in dotted-decimal notation and LAN addresses are shown in hexadecimal notation. Now suppose that the node with IP address 222.222.222.220 wants to send an IP datagram to node 222.222.222.222. To accomplish this task, the sending node must give its adapter not only the IP datagram but also the LAN address for node 222.222.222.222. When passed the IP datagram and the LAN address, the sending node's adapter will construct a data link layer frame containing the receiving node's LAN address and send the frame into the LAN. But how does the sending node determine the LAN address for the node with IP address 222.222.222.222? It does this by providing its ARP module with the IP address 222.222.222.222. ARP then responds with the corresponding LAN address, namely, 49-BD-D2-C7-56-2A.

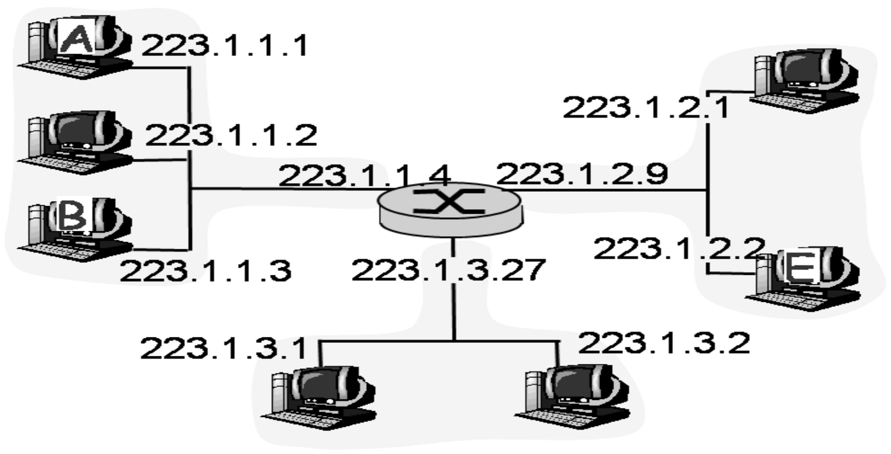


Figure 4-2 Each node on a LAN has an IP address, and each node's adapter has a LAN address

So we see that ARP resolves an IP address to a LAN address. In many ways it is analogous to DNS (studied in unit-I), which resolves hostnames to IP addresses. However, one important difference between the two resolvers is that, DNS resolves hostnames for hosts anywhere in the Internet, whereas ARP resolves IP addresses only for nodes on the same LAN. If a node in California were to try to use ARP to resolve the IP address for a node in Mississippi, ARP would return with an error.

Now that we have explained what ARP does, let's look at how it works. The ARP module in each node has a table in its RAM called an **ARP table.** This table contains the mappings of IP addresses to LAN addresses. Figure 4-3 shows what an ARPJ table in node 222.222.222.220 might look like. For each address mapping the table also contains a time-to-live (TTL) entry, which indicates when the entry will be deleted from the table. Note that the table does not necessarily contain an entry for every node on the LAN; some nodes may have had entries that expired over time whereas other nodes may never have been entered into the table. A typical expiration time for an entry is 20 minutes from when an entry is placed in an ARP table.

IP address	LAN Address	TTL
222.222.222.221	88-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-AB-90-75-B1	13:52:00

Figure 4-3: A possible ARP table in node 222.222.222.220

Now suppose that node 222.222.222.220 wants to send a datagram that is IP addressed to another node on that LAN. The sending node needs to obtain the LAN address of the destination node, given the IP address of that node. This task is easy if the sending node's ARP table has an entry for the destination node. But what if the ARP table doesn't currently have an entry for the destination node? In particular, suppose node 222.222.222.220 wants to send a datagram to node 222.222.222.222. In this case, the sending node uses the ARP protocol to resolve the address. First, the sending node constructs a special packet called an **ARP packet.** An ARP packet has several fields, including the sending and receiving IP and LAN addresses. Both ARP query and response packets have the same format. The purpose of the ARP query packet is to query all the other nodes on the LAN to determine the LAN address corresponding to the IP address that is being resolved.

Returning to our example, node 222.222.222.220 passes an ARP query packet to the adapter along with an indication that the adapter should send the packet to the LAN broadcast address, namely, FF-FF-FF-FF-FF-FF. The adapter encapsulates the ARP packet in a data link frame, uses the broadcast address for the frame's destination address, and transmits the frame into the LAN. Recalling our social security number/postal address analogy, note that an ARP query is equivalent to a person shouting out in a crowded room of cubicles in some company (say, AnyCorp): "What is the social security number of the person whose postal address is Cubicle 13, Room 112, AnyCorp, Palo Alto, CA?" The frame containing the ARP query is received by- all the other adapters on the LAN, and (because of the broadcast address) each adapter passes the ARP packet within the frame up to its hosting node. Each node checks to see if its IP address matches the destination IP address in the ARP packet. The one node with a match sends back to the querying node a response ARP packet with the desired mapping. The querying node (222.222.222.220) can then update its ARP table and send its IP datagram.

There are a couple of interesting things to note about the ARP protocol. First, the query ARP message is sent within a broadcast frame, whereas the response ARP message is sent within a standard frame. Before reading on you should think about why this is so. Second, ARP is plug-and-play; that is, a node's ARP table gets built automatically it doesn't have to be configured by a system administrator; and if a node is disconnected from the LAN, its entry is eventually deleted from the table.

4-3-1 Sending a Datagram to a Node off the LAN

It should now be clear how ARP operates when a node wants to send a datagram to another node *on the same LAN* (that is, on the same IP network, using the terminology of Chapter 4). But now let's look at the more complicated situation when a node on a LAN wants to send a network-layer datagram to a node *off the LAN* (that is, on another IP network). Let us discuss this issue in the context of Figure 4-4, which shows a simple network consisting of two LANs interconnected by a router.

There are several interesting things to note about Figure 4-4. First, there are two types of nodes: hosts and routers. Each host has exactly one IP address and one adapter. But, as discussed in unit-III, a router has an IP address for *each* of its interfaces. Each router interface also has its own ARP module (in the router) and its own adapter. Because the router in Figure 4-4 has two interfaces, it has two IP addresses, two ARP modules, and two adapters. Of course, each adapter in the network has its own LAN address.

Also note that all of the interfaces connected to LAN 1 have addresses of the form 111.111.111 .xxx and all of the interfaces connected to LAN 2 have the form 222.222.222.xxx. Thus, in this example, the first three bytes of the IP address specifies the "network," whereas the last byte specifies the specific interface in the network. In the CIDR notation of Unit-III, LAN 1 has the network address 111.111.111.000/24 and LAN2 has the network address 222.222.222.000/24.

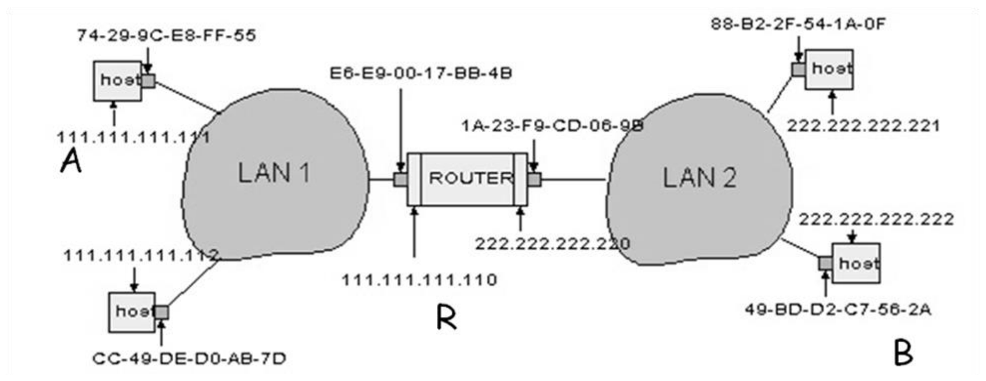


Figure 4-4 Two LANs interconnected by a router

Now suppose that host til.111.111.111 wants to send an IP datagram to host 222.222.222.222. The sending host passes the datagram to its adapter, as usual. But the sending host must also indicate to its adapter an appropriate destination LAN address. What LAN address should the adapter use? One might venture to guess that the appropriate LAN address is that of the adapter for host 222.222.222.222, namely, 49-BD-D2-C7-56-2A. This guess is, however, wrong. If the sending adapter were to use that LAN address, then none of the adapters on LAN 1 would bother to pass the IP datagram up to its network layer, since the frame's destination address would not match the LAN address of any adapter on LAN 1. The datagram would just die and go to datagram heaven.

If we look carefully at Figure 4-4, we see that in order for a datagram to go from 111.111.111.111 to a node on LAN 2, the datagram must first be sent to the router interface 111.111.111.110. As discussed in unit-III, the forwarding table in host 111.111.111.111 would indicate that to reach host 222.222.222.222, the datagram must first be sent to router interface 111.111.111.110. Thus, the appropriate LAN address for the frame is the address of the adapter for router interface 111.111.111.110, namely, E6-E9-00-17-BB-4B. How does the sending host acquire the LAN address of 111.111.111.110? By using ARP, of course! Once the sending adapter has this LAN address, it creates a frame and sends the frame into LAN 1. The router adapter on LAN 1 sees that the data link frame is addressed to it, and therefore passes the frame to the network layer of the router. Hooray! The IP datagram has successfully been moved from source host to the router! But we are not finished. We still have to move the datagram from the router to the destination. The router now has to determine the correct interface on which the datagram is to be forwarded. As discussed in Section 4.4, this is done by consulting a forwarding table in the router. The forwarding table tells the router that the datagram is to be forwarded via router interface 222.222.222.220. This interface then passes the datagram to its adapter, which encapsulates the datagram in a new frame and sends the frame into LAN 2. This time, the destination LAN address of the frame is indeed the LAN address of the ultimate destination. And how does the router obtain this destination LAN address? From ARP, of course!

5. Ethernet

Structure

Objectives

- Discuss about ethernet basics;
- Describe the CSMA/CD;
- Discuss about the ethernet technologies.

5-1 Introduction

Ethernet has pretty much taken over the LAN market. As recently as the 1980s and the early 1990s, Ethernet faced many challenges from other LAN technologies, including token ring, FDDI and ATM. Some of these other technologies succeeded at capturing a part of the market share for a few years. But since its invention in the mid-1970, Ethernet has continued to evolve and grow, and has held on to its dominant market share. Today, Ethernet is by far the most prevalent LAN technology, and is likely to remain so for the foreseeable future. One might say that Ethernet has been to local area networking what the Internet has been to global networking:

There are many reasons for Ethernet's success. First, Ethernet was the first widely-deployed high-speed LAN. Because it was deployed early, network administrators became intimately familiar with Ethernet its wonders and its quirks and were reluctant to switch over to other LAN technologies when they came on the scene. Second, token ring, FDDI and ATM are more complex and expensive than Ethernet, which further discouraged network administrators from switching over. Third, the most compelling reason to switch to another LAN technology (such as FDDI or ATM) was usually the higher data rate of the new technology; however, Ethernet always fought back, producing versions that operated at equal data rates or higher. Switched Ethernet was also introduced in the early 1990s, which further increased its effective data rates. Finally, because Ethernet has been so popular, Ethernet hardware (in particular, network interface cards) has become a commodity and is remarkably cheap. This low cost is also due o the fact that Ethernet's multiple access protocol, CSMA/CD, is totally decentralized, which has also contributed to the low cost and simple design.

The original Ethernet LAN, as shown in Figure 5-1, was invented in the mid 1970s by Bob Metcalfe. An excellent source of online information about Ethernet is Spurgeon's Ethernet Web Site.

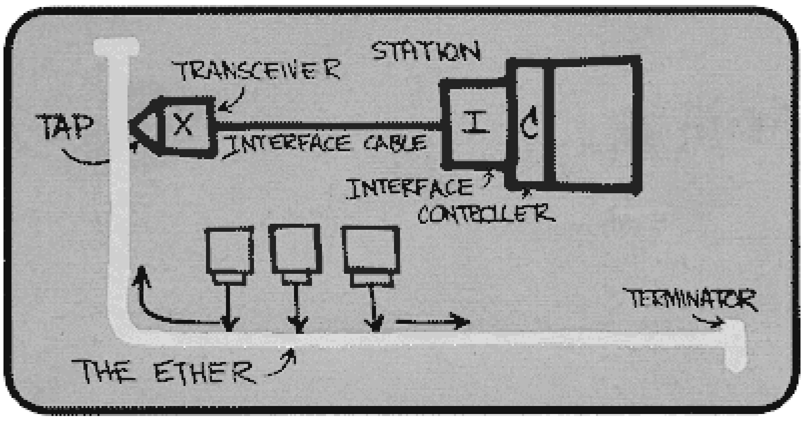


Figure 5-1: The original Metcalfe design led to the 10Base5 Ethernet standard, which included an interface cable that connected the Ethernet adapter (i.e., interface) to an external transceiver.

5-2 Ethernet Basics

Today Ethernet comes in many shapes and forms. An Ethernet LAN can have a "bus topology" or a "star topology." An Ethernet LAN can run over coaxial cable, twisted-pair copper wire, or fiber optics. Furthermore, Ethernet can transmit data at different rates, specifically, at 10 Mbps, 100 Mbps and 1 Gbps. But even though Ethernet comes in many flavors, all of the Ethernet technologies share a few important characteristics. Before examining the different technologies, let's first take a look at the common characteristics.

5-2-1 Ethernet Frame Structure

Given that there are many different Ethernet technologies on the market today, what do they have in common, what binds them together with a common name? First and foremost is the Ethernet frame structure. All of the Ethernet technologies -- whether they use coaxial cable or copper wire, whether they run at 10 Mbps, 100 Mbps or 1 Gbps -- use the same frame structure.

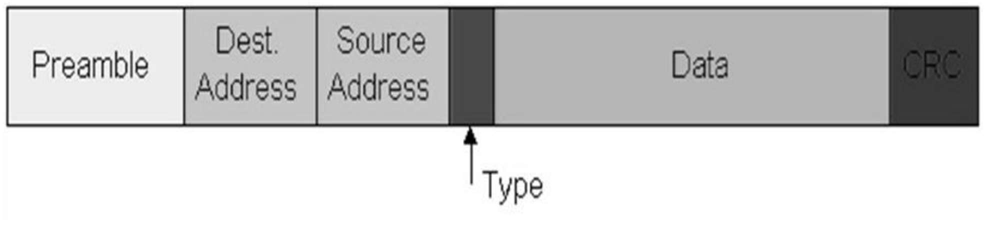


Figure 5-2: Ethernet frame structure

The Ethernet frame is shown in Figure 5-2. Once we understand the Ethernet frame, we will already know a lot about Ethernet. To put our discussion of the Ethernet frame in a tangible context, let us consider sending an IP datagram from one host to another host, with both hosts on the same Ethernet LAN. Let the sending adapter, adapter A, have physical address AA-AA-AA-AA-AA-AA and the receiving adapter, adapter B, have physical address BB-BB-BB-BB-BB-BB. The sending adapter encapsulates the IP datagram within an Ethernet frame and passes the frame to the physical layer. The receiving adapter receives the frame from the physical layer, extracts the IP datagram, and passes the IP datagram to the network layer. In this context, let us now examine the six fields of the Ethernet frame:

- **Data Field** (46 to 1500 bytes): This field carries the IP datagram. The Maximum Transfer Unit (MTU) of Ethernet is 1500 bytes. This means that if the IP datagram exceeds 1500 bytes, then the host has to fragment the datagram, as discussed in Section 4.4. The minimum size of the data field is 46 bytes. This means that if the IP datagram is less than 46 bytes, the data field has to be "stuffed" to fill it out to 46 bytes. When stuffing is used, the data passed to the network layer contains the stuffing as well as an IP datagram. The network layer uses the length field in the IP datagram header to remove the stuffing.
- **Destination Address** (6 bytes): This field contains the LAN address of the destination adapter, namely, BB-BB-BB-BB-BB-BB. When adapter B receives an Ethernet frame with destination address *other* than its own physical address, BB-BB-BB-BB-BB-BB, or the LAN broadcast address, it discards the frame. Otherwise, it passes the contents of the data field to the network layer.
- **Source Address** (6 bytes): This field contains the LAN address of the adapter that transmits the frame onto the LAN, namely, AA-AA-AA-AA-AA-AA.
- **Type Field** (two bytes): The type field permits Ethernet to "multiplex" network-layer protocols. To understand this idea, we need to keep in mind that hosts can use other network-layer protocols besides IP. In fact, a given host may support multiple network layer protocols, and use different protocols for different applications. For this reason, when the Ethernet frame arrives at adapter B, adapter B needs to know to which network-layer protocol it should pass the contents of the data field. IP and other data-link layer protocols (e.g., Novell IPX or AppleTalk) each have there own,

standardized type number. Furthermore, the ARP protocol (discussed in the previous section) has its own type number. Note that the type field is analogous to the protocol field in the network-layer datagram and the port number fields in the transport-layer segment; all of these fields serve to glue a protocol at one layer to a protocol at the layer above.

- **Cyclic Redundancy Check (CRC)** (4 bytes): As discussed in lesson 2, the purpose of the CRC field is to allow the receiving adapter, adapter B, to detect whether any errors have been introduced into the frame, i.e., if bits in the frame have been toggled. Causes of bit errors include attenuation in signal strength and ambient electromagnetic energy that leaks into the Ethernet cables and interface cards. Error detection is performed as follows. When host A constructs the Ethernet frame, it calculates a CRC field, which is obtained from a mapping of the other bits in frame (except for the preamble bits). When host B receives the frame, it applies the same mapping to the frame and checks to see if the result of the mapping is equal to what is in the CRC field. This operation at the receiving host is called the **CRC check**. If the CRC check fails (that is, if the result of the mapping does not equal the contents of the CRC field), then host B knows that there is an error in the frame.
- **Preamble:** (8 bytes) The Ethernet frame begins with an eight-byte preamble field. Each of the first seven bytes of the preamble is 10101010; the last byte is 10101011. The first seven bytes of the preamble serve to "wake up" the receiving adapters and to synchronize their clocks to that of the sender's clock. Why should the clocks be out of synchronization? Keep in mind that adapter A aims to transmit the frame at 10 Mbps, 100 Mbps or 1 Gbps, depending on the type of Ethernet LAN. However, because nothing is absolutely perfect, adapter A will not transmit the frame at exactly the target rate; there will always be some *drift* from the target rate, a drift which is not known *a priori* by the other adapters on the LAN. A receiving adapter can lock onto adapter A's clock by simply locking onto the bits in the first seven bytes of the preamble. The last two bits of the eighth byte of the preamble (the first two consecutive 1s) alert adapter B that the "important stuff" is about to come. When host B sees the two consecutive 1s, it knows that the next six bytes is the destination address. An adapter can tell when a frame ends by simply detecting absence of current.

5-2-2 An Unreliable Connectionless Service

All of the Ethernet technologies provide **connectionless service** to the network layer. That is to say, when adapter A wants to send a datagram to adapter B, adapter A encapsulates the datagram in an Ethernet frame and sends the frame into the LAN, without first "handshaking" with adapter B. This layer-2 connectionless service is analogous to IP's layer-3 datagram service and UDP's layer-4 connectionless service.

All the Ethernet technologies provide an **unreliable service** to the network layer. In particular when adapter B receives a frame from A, adapter B does not send an acknowledgment when a frame passes the CRC check (nor does it send a negative acknowledgment when a frame fails the CRC check). Adapter A hasn't the slightest idea whether a frame arrived correctly or incorrectly. When a frame fails the CRC check, adapter B simply discards the frame. This lack of reliable transport (at the link layer) helps to make Ethernet simple and cheap. But it also means that the stream of datagram's passed to the network layer can have gaps.

If there are gaps due to discarded Ethernet frames, does the application-layer protocol at host B see gaps as well? As we learned in unit-II, this solely depends on whether the application is using UDP or TCP. If the application is using UDP, then the application-layer protocol in host B will indeed suffer from gaps in the data. On the other hand, if the application is using TCP, then TCP in host B will not acknowledge the discarded data, causing TCP in host A to retransmit. Note that when TCP retransmits data, Ethernet retransmits the data as well. But we should keep in mind that Ethernet doesn't know that it is retransmitting. Ethernet thinks it is receiving a brand new datagram with brand new data, even though this datagram contains data that has already been transmitted at least once.

5-2-3 Baseband Transmission and Manchester Encoding

Ethernet uses baseband transmission, that is, the adapter sends a digital signal directly into the broadcast channel. The interface card does not shift the signal into another frequency band, as do ADSL and cable modem systems. With Manchester encoding each bit contains a transition; a 1 has a transition from up to down, whereas a zero has a transition from down to up. The reason for Manchester encoding is that the clocks in the sending and receiving adapters are not perfectly synchronized. By including a transition in the middle of each bit, the receiving host can synchronize its clock to that of the sending host. Once the receiving adapter's clock is synchronized, the receiver can delineate each bit and determine whether it is a one or zero. Manchester encoding is a physical layer operation rather than a link-layer operation; however, we have briefly described it here as it is used extensively in Ethernet.

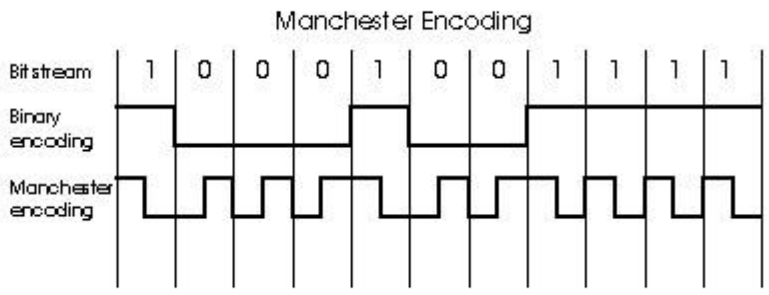


Figure 5-3: Manchester encoding

5-3 CSMA/CD: Ethernet's Multiple Access Protocol

Nodes in an Ethernet LAN are interconnected by a broadcast channel, so that when an adapter transmits a frame, all the adapters on the LAN receive the frame. As we discussed in lesson 3, Ethernet uses a CSMA/CD multiple access algorithm. Summarizing our discussion from lesson 3, recall that CSMA/CD employs the following mechanisms:

1. An adapter may begin to transmit at any time, i.e., no slots are used.
2. An adapter never transmits a frame when it senses that some other adapter is transmitting, i.e., it uses carrier-sensing.
3. A transmitting adapter aborts its transmission as soon as it detects that another adapter is also transmitting, i.e., it uses collision detection.
4. Before attempting a retransmission, an adapter waits a random time that is typically small compared to a frame time.

These mechanisms give CSMA/CD much better performance than slotted ALOHA in a LAN environment. In fact, if the maximum propagation delay between stations is very small, the efficiency of CSMA/CD can approach 100%. But note that the second and third mechanisms listed above require each Ethernet adapter to be able to (1) sense when some other adapter is transmitting, and (2) detect a collision while it is transmitting. Ethernet adapters perform these two tasks by measuring voltage levels before and during transmission.

Each adapter runs the CSMA/CD protocol without explicit coordination with the other adapters on the Ethernet. Within a specific adapter, the CSMA/CD protocol works as follows:

1. The adapter obtains a network-layer PDU from its parent node, prepares an Ethernet frame, and puts the frame in an adapter buffer.
2. If the adapter senses that the channel is idle (i.e., there is no signal energy from the channel entering the adapter), it starts to transmit the frame. If the adapter senses that the channel is busy, it waits until it senses no signal energy (plus a few hundred microseconds) and then starts to transmit the frame.
3. While transmitting, the adapter monitors for the presence of signal energy coming from other adapters. If the adapter transmits the entire frame without detecting signal energy from other adapters, the adapter is done with the frame.
4. If the adapter detects signal energy from other adapters while transmitting, it stops transmitting its frame and instead transmits a 48-bit jam signal.

5. After aborting (i.e., transmitting the jam signal), the adapter enters an **exponential backoff** phase. Specifically, when transmitting a given frame, after experiencing the *n*th collision in a row for this frame, the adapter chooses a value for *K* at random from $\{0,1,2,...,2^m - 1\}$ where $m := min(n,10)$. The adapter then waits *K* x 512 bit times and then returns to Step 2.

A few comments about the CSMA/CD protocol are certainly in order. The purpose of the jam signal is to make sure that all other transmitting adapters become aware of the collision. Let's look at an example. Suppose adapter A begins to transmit a frame, and just before A's signal reaches adapter B, adapter B begins to transmit. So B will have transmitted only a few bits when it aborts its transmission. These few bits will indeed propagate to A, but they may not constitute enough energy for A to detect the collision. To make sure that A detects the collision (so that it to can also abort), B transmits the 48-bit jam signal.

Next consider the exponential backoff algorithm. The first thing to notice here is that a bit time (i.e., the time to transmit a single bit) is very short; for a 10 Mbps Ethernet, a bit time is .1 microseconds. Now let's look at an example. Suppose that an adapter attempts for the first time to transmit a frame, and while transmitting it detects a collision. The adapter then chooses *K=0* with probability .5 and chooses *K=1* with probability .5. If the adapter chooses *K=0*, then it immediately jumps to Step 2 after transmitting the jam signal. If the adapter chooses *K=1*, it waits 51.2 microseconds before returning to Step 2. After a second collision, *K* is chosen with equal probability from {0,1,2,3}. After three collisions, *K* is chosen with equal probability from {0,1,2,3,4,5,6,7}. After ten or more collisions, *K* is chosen with equal probability from {0,1,2,...,1023}. Thus the size of the sets from which *K* is chosen grows exponentially with the number of collisions (until *n=10*); it is for this reason that Ethernet's backoff algorithm is referred to as "exponential backoff".

The Ethernet standard imposes limits on the distance between any two nodes. These limits ensure that if adapter A chooses a lower value of *K* than all the other adapters involved in a collision, then adapter A will be able to transmit its frame without experiencing a new collision. We will explore this property in more detail in the homework problems.

Why use exponential backoff? Why not, for example, select *K* from {0,1,2,3,4,5,6,7} after every collision? The reason is that when an adapter experiences its first collision, it has no idea how many adapters are involved in the collision. If there are only a small number of colliding adapters, it makes sense to choose *K* from a small set of small values. On the other hand, if many adapters are involved in the collision, it makes sense to choose *K* from a larger, more dispersed set of values (why?). By increasing the size of the set after each collision, the adapter appropriately adapts to these different scenarios.

We also note here that each time an adapter prepares a new frame for transmission; it runs the CSMA/CD algorithm presented above. In particular, the adapter does not take into account any collisions that may have occurred in the recent past. So it is possible that an adapter with a new frame will be able to immediately sneak in a successful transmission while several other adapters are in the exponential backoff state.

5-3-1 Ethernet Efficiency

When only one node has a frame to send (which is typically the case), the node can transmit at the full rate of the Ethernet technology (10 Mbps, 100 Mbps, or 1 Gbps). However, if many nodes have frames to transmit, the effective transmission rate of the channel can be much less. We define the **efficiency of Ethernet** to be the long-run fraction of time during which frames are being transmitted on the channel without collisions when there is a large number of active nodes, with each node having a large number of frames to send. In order to present a closed-form approximation of the efficiency of Ethernet, let t_{prop} denote the maximum time it takes signal energy to propagate between any two adapters. Let t_{trans} be the time to transmit a maximum size Ethernet frame (approximately 1.2 msecs for a 10 Mbps Ethernet). Here we simply state the following approximation:

$$\text{Efficiency} = 1/(1+ 5\ t_{prop}/t_{trans}).$$

We see from this formula that as t_{prop} approaches 0, the efficiency approaches 1. This is intuitive because if the propagation delay is zero, colliding nodes will abort immediately without wasting the channel. Also, as t_{trans} becomes very large, efficiency approaches 1. This is also intuitive because when a frame grabs the channel, it will hold on to the channel for a very long time; thus the channel will be doing productive work most of the time.

5-4 Ethernet Technologies

The most common Ethernet technologies today are 10Base2, which uses thin coaxial cable in a bus topology and has a transmission rate of 10 Mbps; 10BaseT, which uses twisted-pair cooper wire in a star topology and has a transmission rate of 10 Mbps; 100BaseT, which typically uses twisted-pair cooper wire in a star topology and has a transmission rate of 100 Mbps; and Gigabit Ethernet, which uses both fiber and twisted-pair cooper wire and transmits at a rate of 1 Gbps. These Ethernet technologies are standardized by the IEEE 802.3 working groups. For this reason, Ethernet is often referred to as an 802.3 LAN.

Before discussing specific Ethernet technologies, we need to discuss **repeaters**, which are commonly used in LANs as well as in wide-area transport. A repeater is a physical-layer device that acts on individual bits rather than on packets. It has two or more interfaces. When a bit, representing a zero or a one, arrives from one interface, the repeater simply recreates the bit, boosts its energy strength, and transmits the bit onto all the other interfaces. Repeaters are commonly used in LANs in order to extend their geographical range. When used with Ethernet, it is important to keep in mind that repeaters do not implement carrier sensing or any other part of CSMA/CD; a repeater repeats an incoming bit on all outgoing interfaces even if there is signal energy on some of the interfaces.

5-4-1 10Base2 Ethernet

10Base2 is a very popular Ethernet technology. If you look at how your computer (at work or at school) is connected to the network, it is very possible you will see a 10Base2 connection. The "10" in 10Base2 stands for "10 Mbps"; the "2" stands for "200 meters", which is the approximate maximum distance between any two nodes without repeaters between them. (The actual maximum distance is 185 meters.) A 10Base2 Ethernet is shown in Figure 5-4.

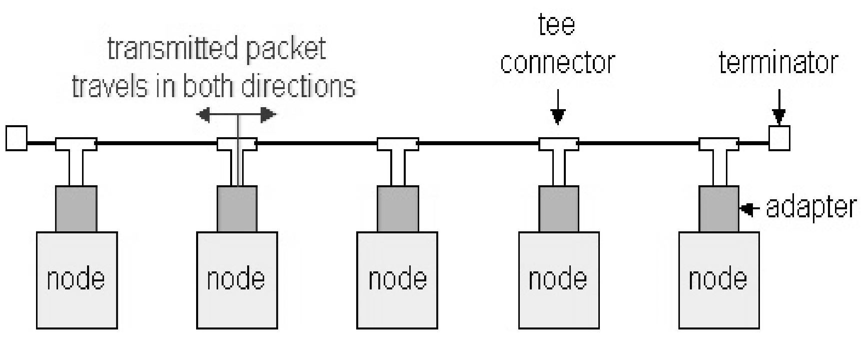


Figure 5-4: A 10Base2 Ethernet

We see from Figure 4.3 in lesson-4 that 10Base2 uses a bus topology; that is, nodes are connected (through their adapters) in a linear fashion. The physical medium used to connect the nodes is **thin coaxial cable**, which is similar to what is used in cable TV, but with a thinner and lighter cable. When an adapter transmits a frame, the frame passes through a "tee connector;" two copies of the frame leave the tee connector, one copy going in one direction and one copy in the other direction. As the frames travel towards the terminators, they leave a copy at every node they pass. (More precisely, as a bit passes in front of a node, part of the energy of the bit leaks into the adapter.) When the frame finally reaches a terminator, it gets absorbed by the terminator. Note when an adapter transmits a frame, the frame is received by every other adapter on the Ethernet. Thus, 10Base2 is indeed a broadcast technology.

Suppose you want to connect a dozen PCs in your office using 10Base2 Ethernet. To do this, you would need to purchase 12 Ethernet cards with thin Ethernet ports; 12 BNC trees, which are small metallic objects that attach to the adapters (less than one dollar each); a dozen or so thin coax segments, 5-20 meters each; and two "terminators," which you put at the two ends of the bus. The cost of the whole network, including adapters, is likely to be less than the cost of a single PC! Because 10Base2 is incredibly inexpensive, it is often referred to as "cheapnet".

Without a repeater, the maximum length of a 10Base2 bus is 185 meters. If the bus becomes any longer, then signal attenuation can cause the system to malfunction. Also, without a repeater, the maximum number of nodes is 30, as each node contributes to signal attenuation. Repeaters can be used to connect 10Base2 segments in a linear fashion, with each segment having up to 30 nodes and having a length up to 185 meters. Up to four repeaters can be included in a 10Base2 Ethernet, which creates up to five "segments". Thus a 10Base2 Ethernet bus can have a total length of 985 meters and support up to 150 nodes. Note that the CSMA/CD access protocol is completely oblivious to the repeaters; if any two of 150 nodes transmit at the same time, there will be a collision.

5-4-2 10BaseT and 100BaseT

We discuss 10BaseT and100BaseT Ethernet together, as they are similar technologies. The most important difference between them is that 10BaseT transmits at 10 Mbps and 100BaseT Ethernet transmits at 100 Mbps. 100BaseT is also commonly called "fast Ethernet" and "100 Mbps Ethernet". 10BaseT and 100BaseT are also very popular Ethernet technologies; in fact, for new installations, 10BaseT and Ethernet are often today the technology of choice. Both 10BaseT and 100BaseT Ethernet use a star topology, as shown in Figure 5-5.

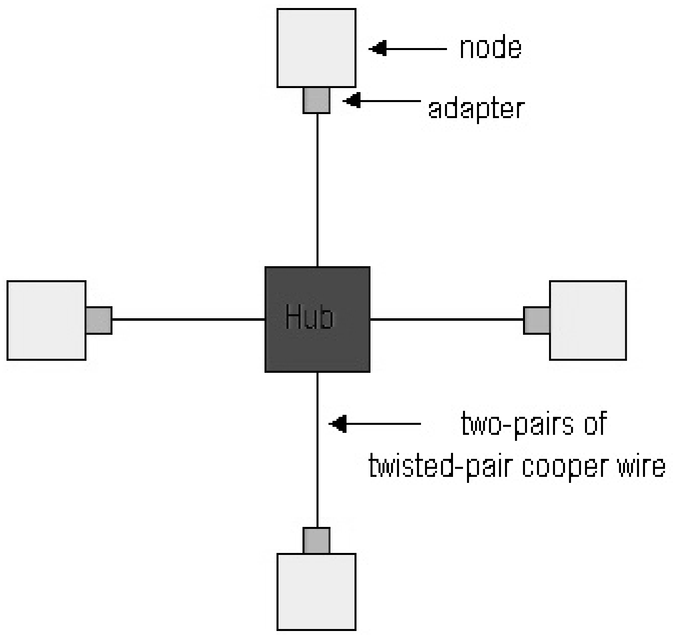


Figure 5-5: Star topology for 10BaseT and 100BaseT

In the star topology there is a central device called a **hub** (also sometimes called a concentrator.) Each adapter on each node has a direct, point-to-point connection to the hub. This connection consists of two pairs of twisted-pair cooper wire, one for transmitting and the other for receiving. At each end of the connection there is a connector that resembles the RJ-45 connector used for ordinary telephones. The "T" in 10BaseT and 100BaseT stands for "twisted pair". For both 10BaseT and 100BaseT, the maximum length of the connection between an adapter and the hub is 100 meters; the maximum length between any two nodes is 200 meters.

In essence, a hub is a repeater: when it receives a bit from an adapter, it sends the bit to all the other adapters. In this manner, each adapter can (1) sense the channel to determine if it is idle, and (2) detect a collision while it is transmitting. But hubs are popular because they also provide network management features. For example, if an adapter malfunctions and continually sends Ethernet frames (a so-called "jabbering adapter"), then in a 10Base2 Ethernet will become totally dysfunctional; none of the nodes will be able to communicate. But a 10BaseT network will continue to function, because the hub will detect the problem and internally disconnect the malfunctioning adapter. With this feature, the network administrator doesn't have to get out of bed and drive back to work in order to correct the problem for hackers who work late at night. Also, most hubs can gather information and report the information to a host that connects directly to the hub. This monitoring host provides a graphical interface that displays statistics and graphs, such as bandwidth usage, collision rates, average frame sizes, etc. Network administrators can use this information to not only debug and correct problems, but also to plan how the LAN should evolve in the future.

Many Ethernet adapters today are 10/100 Mbps adapters. This means that they can be used for both 10BaseT and 100BaseT Ethernets. 100BaseT, which typically uses category-5 twisted pair (a high-quality twisted pair with a lot of twists). Unlike the 10Base2 and 10BaseT, 100BaseT does not use Manchester encoding, but instead a more efficient encoding called 4B5B: every group of five clock periods is used to send 4 bits in order to provide enough transitions to allow clock synchronization.

We briefly mention at this point that both 10 Mbps and 100 Mbps Ethernet technologies can employ fiber links. A fiber link is often used to interconnect to hubs that are in different buildings on the same campus. Fiber is expensive because of cost of the cost of its connectors, but it has excellent noise immunity. The IEEE 802 standards permit a LAN to have a larger geographically reach when fiber is used to connect backbone nodes.

5-4-3 Gigabit Ethernet

Gigabit Ethernet is an extension to the highly successful 10 Mbps and 100 Mbps Ethernet standards. Offering a raw data rate of 1000 Mbps, Gigabit Ethernet maintains full compatibility with the huge installed base of Ethernet equipment. The standard for Gigabit Ethernet, referred to as IEEE 802.3z, does the following:

- Uses the standard Ethernet frame format, and is backward compatible with 10BaseT and 100BaseT technologies. This allows for easy integration of Gigabit Ethernet with the existing installed base of Ethernet equipment.
- Allows for point-to-point links as well as shared broadcast channels. Point-to-point links use switches (see in lesson 6) where as broadcast channels use hubs, as described above for 10BaseT and 100 Base-T. Un Gigabit Ethernet jargon, hubs are called "buffered distributors".
- Uses CSMA/CD for shared broadcast channels. In order to have acceptable efficiency, the maximum distance between nodes must be severely restricted.
- Allows for full-duplex operation at 1000 Mbps in both directions for point-to-point channels.

Like 10BaseT and 100BaseT, Gigabit Ethernet has a star topology with a hub or switch at its center. (Ethernet switches will be discussed in lesson 6.) Gigabit Ethernet often serves as a backbone for interconnecting multiple 10 Mbps and 100 Mbps Ethernet LANs. Initially operating over optical fiber, Gigabit Ethernet will be able to use Category 5 UTP cabling.

The Gigabit Ethernet Alliance is an open forum whose purpose is to promote industry cooperation in the development of Gigabit Ethernet.

6. Bridges and Switches

Structure

Objectives

- Discuss about intera network devices;
- Discuss about cut-through switch.

6-1 Introduction

Institutions including, companies, universities and high schools typically consist of many departments, with each department having and managing its own Ethernet LAN. Naturally, an institution will want its departments to interconnect their departmental LAN segments. In this section, we consider a number of different approaches in which LANs can be connected together. We'll cover three approaches, hubs, bridges, and switches in the following subsections. All three of these approaches are in widespread use today.

6-2 Hubs

The simplest way to interconnect LANs is to use a hub. A hub is a simple device that takes an input (i.e., a frame's bits) a retransmits the input on the hub's outgoing ports. Hubs are essentially repeaters, operating on bits. They are thus physical-layer devices. When a bit comes into a hub interface, the hub simply broadcasts the bit on all the other interfaces. In this section we investigate bridges, which are another type of interconnection device.

Figure 6-1 shows how three academic departments in a university might interconnect their LANs. In this figure, each of the three departments has a 10BaseT Ethernet that provides network access to the faculty, staff and students of the departments. Each host in a department has a point-to-point connection to the departmental hub. A fourth hub, called a **backbone hub**, has point-to-point connections to the departmental hubs, interconnecting the LANs of the three departments. The design shown in Figure 6-1 is a **multi-tier hub design** because the hubs are arranged in a hierarchy. It is also possible to create multi-tier designs with more than two tiers -- for example, one tier for the departments, one tier for the schools within the university (e.g., engineering school, business school, etc.) and one tier at the highest university level. Multiple tiers can also be created out of 10Base2 (bus topology Ethernets) with repeaters.

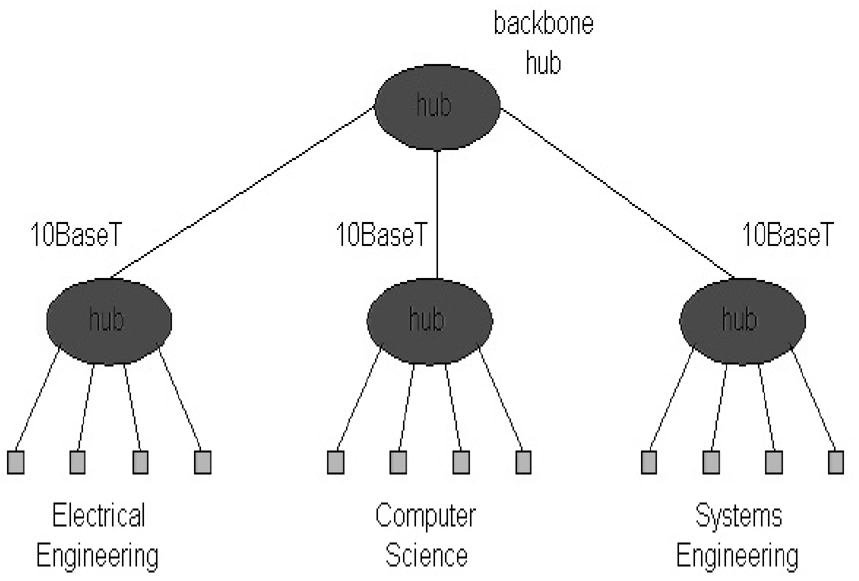


Figure 6-1: Three departmental Ethernets interconnected with a hub.

In a multi-tier design, we refer to the entire interconnected network as a LAN, and we refer to each of the departmental portions of the LAN (i.e., the departmental hub and the hosts that connect to the hub) as a **LAN segment**. It is important to note that all of the LAN segments in Figure 6-1 belong to the same **collision domain**, that is, whenever two or more nodes on the LAN segments transmit at the same time, there will be a collision and all of the transmitting nodes will enter exponential backoff.

Interconnecting departmental LANs with a backbone hub has many benefits. First and foremost, it provides inter-departmental communication to the hosts in the various departments. Second, it extends the maximum distance between any pair of nodes on the LAN. For example, with 10BaseT the maximum distance between a node and its hub is 100 meters; therefore, in a single LAN segment the maximum distance between any pair of nodes is 200 meters. By interconnecting the hubs, this maximum distance can be extended, since the distance between directly-connected hubs can also be 100 meters when using twisted pair (and more when using fiber). Third, the multi-tier design provides a degree of graceful degradation. Specifically, if any one of the departmental hubs starts to malfunction, the backbone hub can detect the problem and disconnect the departmental hub from the LAN; in this manner, the remaining departments can continue to operate and communicate while the faulty departmental hub gets repaired.

Although a backbone hub is a useful interconnection device, it has three serious limitations that hinder its deployment. First, and perhaps more important, when departmental LANs are interconnected with a hub (or a repeater), then the independent collision domains of the departments are transformed into one large and common collision domain. Let us explore this latter issue in the context of Figure 6-1. Before interconnecting the three departments, each departmental LAN had a maximum throughput of 10 Mbps, so that maximum aggregate throughput of the three LANs was 30 Mbps. But once the three LANs are interconnected with a hub, all of the hosts in the three departments belong to the same collision domain, and the maximum aggregate throughput is reduced to 10 Mbps.

A second limitation is that if the various departments use different Ethernet technologies, then it may not be possible to interconnect the departmental hubs with a backbone hub. For example, if some department's use 10BaseT and the remaining department's use 100BaseT, then it is impossible to interconnect all the departments without some frame buffering at the interconnection point; since hubs are essentially repeaters and do not buffer frames, they cannot interconnect LAN segments operating at different rates.

A third limitation is that each of the Ethernet technologies (10Base2, 10BaseT, 100BaseT, etc.) has restrictions on the maximum number of nodes that can be in a collision domain, the maximum distance between two hosts in a collision domain, and the maximum number of tiers that can be present in a multi-tier design. These restrictions constrain both the total number of hosts that connect to a multi-tier LAN as well as geographical reach of the multi-tier LAN.

6-3 Bridges

In contrast to hubs, which are physical-level devices, bridges operate on Ethernet frames and thus are layer-2 devices. In fact, bridges are full-fledged packet switches that forward and filter frames using the LAN destination addresses. When a frame comes into a bridge interface, the bridge does not just copy the frame onto all of the other interfaces. Instead, the bridge examines the destination address of the frame and attempts to forward the frame on the interface that leads to the destination.

Figure 6-2 shows how the three academic departments of our previous example might be interconnected with a bridge. The three numbers next to the bridge are the interface numbers for the three bridge interfaces. When the departments are interconnected by a bridge, as in Figure 6-2, we again refer to the entire interconnected network as a LAN, and we again refer to each of the departmental portions of the network as LAN segments. But in contrast to the multi-tier hub design in Figure 5.6-1, each LAN segment is now an isolated collision domain.

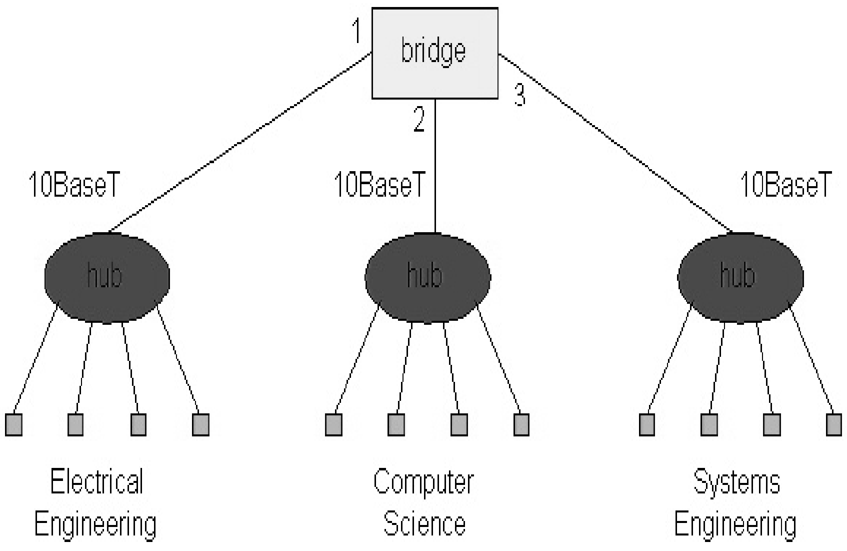


Figure 6-2: Three departmental LANs interconnected with a bridge.

Bridges can overcome many of the problems that plague hubs. First, bridges permit inter-departmental communication while preserving isolated collision domains for each of the departments. Second, bridges can interconnect different LAN technologies, including 10 Mbps and 100 Mbps Ethernets. Third, there is no limit to how big a LAN can be when bridges are used to interconnect LAN segments: in theory, using bridges, it is possible to build a LAN that spans the entire globe.

6-3-1 Bridge Forwarding and Filtering

Filtering is the ability to determine whether a frame should be forwarded to an interface or should just be dropped. When the frame should be forwarded, **forwarding** is the ability to determine which of the interfaces the frame should be directed to. Bridge filtering and forwarding are done with a **bridge table**. For each node on the LAN, the bridge table contains (1) the LAN address of the node, (2) the bridge interface that leads towards the node, (3) and the time at which the entry for the node was placed in the table. An example Table for the LAN in Figure 6-2 is shown in Figure 6-3. We note here that the addressees used by bridges are physical addresses (not network addresses). We will also see shortly that a bridge table is constructed in a very different manner than routing tables.

Address	Interface	Time
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
...	...	...

Figure 6-3: Portion of a bridge table for the LAN in Figure 6-2.

To understand how bridge filtering and forwarding works, suppose a frame with destination address DD-DD-DD-DD-DD-DD arrives to the bridge on interface x. The bridge indexes its table with the LAN address DD-DD-DD-DD-DD-DD and finds the corresponding interface y.

- If x equals y, then the frame is coming from a LAN segment that contains adapter DD-DD-DD-DD-DD-DD. There being no need to forward the frame to any of the other interfaces, the bridge performs the filtering function by discarding the frame.
- If x does not equal y, then the frame needs to be routed to the LAN segment attached to interface y. The bridge performs its forwarding function by putting the frame in an output buffer that precedes interface y.

These simple rules allow a bridge to preserve separate collision domains for each of the different LAN segments connected to its interfaces. The rules also allow the nodes on different LAN segments to communicate.

Let's walk through these rules for the network in Figures 6-2 and its bridge table in Figure 6-3. Suppose that a frame with destination address 62-FE-F7-11-89-A3 arrives to the bridge from interface 1. The bridge examines its table and sees that the destination is on the LAN segment connected to interface 1 (i.e., the Electrical Engineering LAN). This means that the frame has already been broadcast on the LAN segment that contains the destination. The bridge therefore filters (i.e., discards) the frame. Now suppose a frame with the same destination address arrives from interface 2. The bridge again examines its table and sees that the destination is the direction of interface 1; it therefore forwards the frame to the output buffer preceding interface 1. It should be clear from this example that as long as the bridge table is complete and accurate, the bridge isolates the departmental collision domains while permitting the departments to communicate.

Recall that when a hub (or a repeater) forwards a frame onto a link, it just sends the bits onto the link without bothering to sense whether another transmission is currently taking place on the link. In contrast, when a bridge wants to forward a frame onto a link, it runs the CSMA/CD algorithm discussed in unit-III. In particular, the bridge refrains from transmitting if it senses that some other node on the LAN segment is transmitting; furthermore, the bridge uses exponential backoff when one of its transmissions results in a collision. Thus bridge interfaces behave very much like node adapters. But technically speaking, they are *not* node adapters because neither a bridge nor its interfaces have LAN addresses. Recall that a node adapter always inserts its LAN address into the source address of every frame it transmits. This statement is true for router adapters as well as host adapters. A bridge, on the other hand, does not change the source address of the frame.

One significant feature of bridges is that they can be used to combine Ethernet segments using different Ethernet technologies. For example, if in Figure 6-2, Electrical Engineering has a 10Base2 Ethernet, Computer Science has a 100BaseT Ethernet, and Electrical Engineering has a 10BaseT Ethernet, then a bridge can be purchased that can interconnect the three LANs. With Gigabit Ethernet bridges, it is possible to have an additional 1 Gbps connection to a router, which in turn connects to a larger university network. As we mentioned earlier, this feature of being able to interconnect different link rates is not available with hubs.

Also, when bridges are used as interconnection devices, there is no theoretical limit to the geographical reach of a LAN. In theory, we can build a LAN that spans the globe by interconnecting hubs in a long, linear topology, with each pair of neighboring hubs interconnected by a bridge. Because in this design each of the hubs has its own collision domain, there is no limit on how long the LAN can be. We shall see shortly, however, that it is undesirable to build very large networks exclusively using bridges as interconnection devices large networks need routers as well.

6-3-2 Self-Learning

A bridge has the *very cool* property of building its table automatically, dynamically and autonomously without any intervention from a network administrator or from a configuration protocol. In other words, bridges are **self-learning**. This is accomplished as follows.

- The bridge table is initially empty.
- When a frame arrives on one of the interfaces and the frame's destination address is not in the table, then the bridge forwards copies of the frame to the output buffers of all of the other interfaces. (At each of these other interfaces, the frame accesses the LAN segment using CSMA/CD.)
- For each frame received, the bridge stores in its table (1) the LAN address in the frame's *source address field*, (2) the interface from which the frame arrived, (3) the current time. In this manner the bridge records in its table the LAN segment on which the sending node resides. If every node in the LAN eventually sends a frame, then every node will eventually get recorded in the table.

- When a frame arrives on one of the interfaces and the frame's destination address is in the table, then the bridge forwards the frame to the appropriate interface.
- The bridge deletes an address in the table if no frames are received with that address as the source address after a period of time (the *aging time*). In this manner, if a PC is replaced by another PC (with a different adapter), the LAN address of the original PC will eventually be purged from the bridge table.

Let's walk through the self-learning property for the network in Figures 6-2 and its corresponding bridge table in Figure 6-3. Suppose at time 9:39 a frame with source address 01-12-23-34-45-56 arrives from interface 2. Suppose that this address is not in the bridge table. Then the bridge appends a new entry in the table, as shown in Figure 6-4.

Address	**Interface**	**Time**
01-12-23-34-45-56	2	9:39
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
.....	.....	.....

Figure 6-4: Bridge learns about the location of adapter with address

01-12-23-34-45-56.

Continuing with this same example, suppose that the aging time for this bridge is 60 minutes and no frames with source address 62-FE-F7-11-89-A3 arrive to the bridge between 9:32 and 10:32. Then at time 10:32 the bridge removes this address from its table.

Bridges are **plug and play devices** because they require absolutely no intervention from a network administrator or user. When a network administrator wants to install a bridge, it does no more than connect the LAN segments to the bridge interfaces. The administrator does not have to configure the bridge tables at the time of installation or when a host is removed from one of the LAN segments. Because bridges are plug-and-play, they are also referred as **transparent bridges.**

6-3-3 Spanning Tree

One of the problems with a pure hierarchical design for interconnected LAN segments is that if a hub or a bridge near the top of the hierarchy fails, then much (if not all) of the interconnected LAN will go down. For this reason it is desirable to build networks with multiple paths between LAN segments. An example of such a network is shown in Figure 6-5.

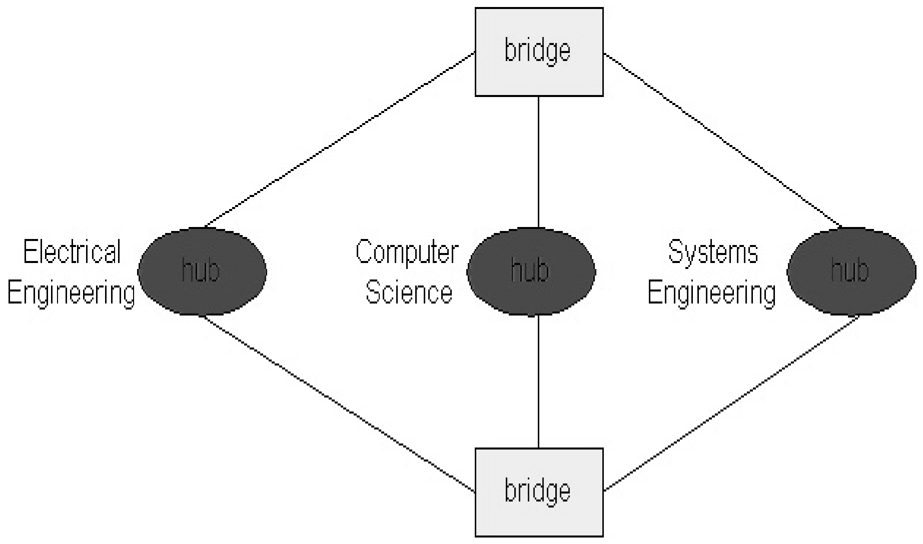


Figure 6-5: Interconnected LAN segments with redundant paths.

Multiple redundant paths between LAN segments (such as departmental LANs) can greatly improve fault tolerance. But, unfortunately, multiple paths have a serious side effect frames cycle and multiply within the interconnected LAN, thereby crashing the entire network. To see this, suppose that the bridge tables in Figure 6-5 are empty, and a host in Electrical Engineering sends a frame to a host in Computer Science. When the frame arrives to the Electrical Engineering hub, the hub will generate two copies of the frame and send one copy to each of the two bridges. When a bridge receives the frame, it will generate two copies; send one copy to the Computer Science hub and the other copy to the Systems Engineering hub. Since both bridges do this, there will be four identical frames in the LAN. This multiplying of copies will continue indefinitely since the bridges do not know where the destination host resides. (To route the frame to the destination host in Computer Science, the destination host has to first generate a frame so that its address can be recorded in the bridge tables.) The number of copies of the original frame grows exponentially fast, crashing the entire network.

To prevent the cycling and multiplying of frames, bridges use a spanning tree protocol. In the **spanning tree protocol**, bridges communicate with each other over the LANs in order to determine a spanning tree, that is, a subset of the original topology that has no loops. Once the bridges determine a spanning tree, the bridges disconnect appropriate interfaces in order to create the spanning tree out of the original topology. For example, in Figure 6-5, a spanning tree is created by having the top bridge disconnect its interface to Electrical Engineering and the bottom bridge disconnects its interface to Systems Engineering. With the interfaces disconnected and the loops removed, frames will no longer cycle and multiply. If, at some later time, one of links in the spanning tree fails, the bridges can reconnect the interfaces, run the spanning tree algorithm again, and determine a new set of interfaces that should be disconnected.

6-3-4 Bridges versus Routers

As we learned in unit-III, routers are store-and-forward packet switches that forward packets using IP addresses. Although a bridge is also a store-and-forward packet switch, it is fundamentally different from a router in that it forwards packets using LAN addresses. Whereas a router is layer-3 packet switch, a bridge is a layer-2 packet switch.

Even though bridges and routers are fundamentally different, network administrators must often choose between them when installing an interconnection device. For example, for the network in Figure 6-2, the network administrator could have just as easily used a router instead of a bridge. Indeed, a router would have also kept the three collision domains separate while permitting interdepartmental communication. Given that both bridges and routers are candidates for interconnection devices, what are the pros and cons of the two approaches?

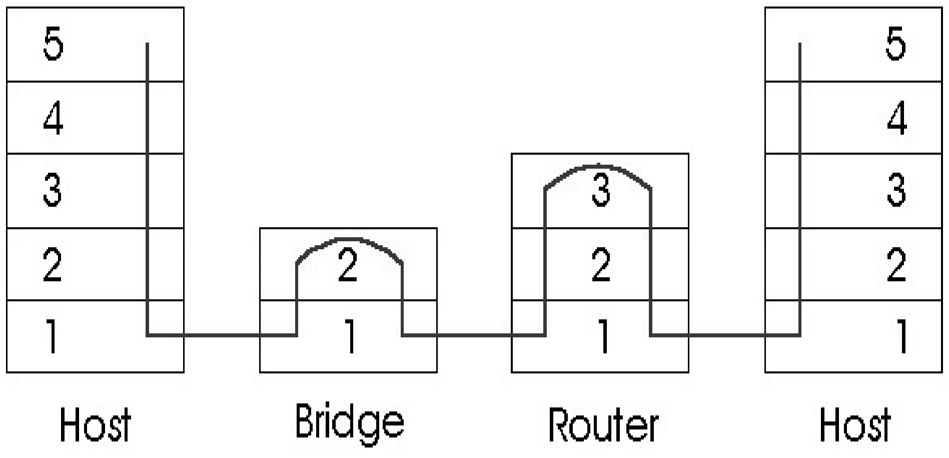


Figure 6-6: Packet processing and bridges, routers and hosts.

First consider the pros and cons of bridges. As mentioned above, bridges are plug-and-play, a property that is cherished by all the over-worked network administrators of the world. Bridges can also have relatively high packet filtering and forwarding rates as shown in Figure 6-6, bridges only have to process packets up through layer 2, whereas routers have to process frames up through layer 3. On the other hand, the spanning tree protocol restricts the effective topology of a bridged network to a spanning tree. This means that all frames most flow along the spanning tree, even when there are more direct (but disconnected) paths between source and destination. The spanning tree restriction also concentrates the traffic on the spanning tree links when it could have otherwise been spread through all the links of the original topology. Furthermore, bridges do not offer any protection against broadcast storms if one host goes haywire and transmits an endless stream of Ethernet broadcast packets, the bridges will forward all of the packets and the entire network will collapse.

Now consider the pros and cons of routers. Because IP addressing is hierarchical (and not flat as is LAN addressing), packets do not normally cycle through routers even when the network has redundant paths. (Actually, packets can cycle when router tables are misconfigured; but as we learned in unit-III, IP uses a special datagram header field to limit the cycling.) Thus, packets are not restricted to a spanning tree and can use the best path between source and destination. Because routers do not have the spanning tree restriction, routers have allowed the Internet to be built with a rich topology which includes, for example, multiple active links between Europe and North America. Another feature of routers is that they provide firewall protection against layer-2 broadcast storms. Perhaps the most significant drawback of routers is that they are not plug-and-play they and the hosts that connect to them need their IP addresses to be configured. Also, routers often have a larger prepackage processing time than bridges, because they have to process up through the layer-3 fields. Finally, there are two different ways to pronounce the word "router", either as "rootor" or as "rowter", and people waste a lot of time arguing over the proper pronunciation.

Given that both bridges and routers have their pros and cons, when should an institutional network (e.g., university campus network or a corporate campus network) use bridges, and when should it use bridges? Typically, small networks consisting of a few hundred hosts have a few LAN segments. Bridges suffice for these small networks, as they localize traffic and increase aggregate throughput without requiring any configuration of IP addresses. But larger networks consisting of thousands of hosts typically include routers within the network (in addition to bridges). The routers provide a more robust isolation of traffic, control broadcast storms, and use more "intelligent" routes among the hosts in the network.

6-3-5 Connecting LAN Segments with Backbones

Consider once again the problem of interconnecting with bridges the Ethernets in the three departments in Figure 6-2. An alternative design is shown in Figure 6-7. This alternative design uses two two-interface bridges (i.e., bridges with two interfaces), with one bridge connecting Electrical Engineering to Computer Science, and the other bridge connecting Computer Science to Systems Engineering. Although two-interface bridges are very popular due to their low cost and simplicity, the design in Figure 6-7 is *not recommended* for two reasons. First, if the Computer Science hub were to fail, then Electrical Engineering and Systems Engineering would no longer be able to communicate. Second, and more important, all the inter-departmental traffic between Electrical and Systems Engineering has to pass through Computer Science, which may overly burden the Computer Science LAN segment.

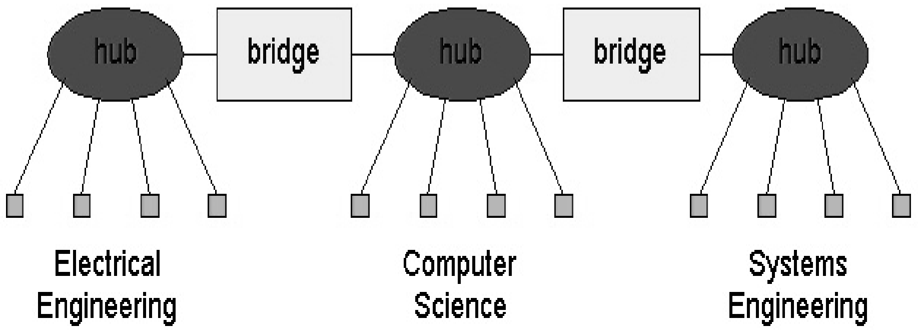


Figure 6-7: An example of an institutional LAN *without* a backbone.

One important principle when designing an interconnected LAN is that the various LAN segments should be interconnected with a backbone. A **backbone** is a network that has direct connections to all the LAN segments. When a LAN has a backbone, then each pair of LAN segments can communicate without passing through a third-party LAN segment. The design shown Figure 6-2 uses a three-interface bridge for a backbone. In the homework problems at the end of this chapter we shall explore how to design backbone networks with two-interface bridges.

6-4 Switches

Up until the mid 1990s, three types of LAN interconnection devices were essentially available: hubs (and their cousins, repeaters), bridges and routers. More recently yet another interconnection device became widely available, namely, Ethernet switches. Ethernet switches, often trumpeted by network equipment manufacturers with great fanfare, are in essence high-performance multi-interface bridges. As do bridges, they forward and filter frames using LAN destination addresses, and they automatically build routing tables using the source addresses in the traversing frames. The most important difference between a bridge and switch is that bridges usually have a small number of interfaces (i.e., 2-4), whereas switches may have dozens of interfaces. A large number interfaces generates a high aggregate forwarding rate through the switch fabric, therefore necessitating a high-performance design (especially for 100 Mbps and 1 Gbps interfaces).

Switches can be purchased with various combinations of 10 Mbps, 100 Mbps and 1 Gbps interfaces. For example, you can purchase switches with four 100 Mbps interfaces and twenty 10 Mbps interfaces; or switches with four 100 Mbps interfaces and one 1 Gbps interface. Of course, the more the interfaces and the higher transmission rates of the various interfaces, the more you pay. Many switches also operate in a **full-duplex mode**; that is, they can send and receive frames at the same time over the same interface. With a full duplex switch (and corresponding full duplex Ethernet adapters in the hosts), host A can send a file to host B while that host B simultaneously sends to host A.

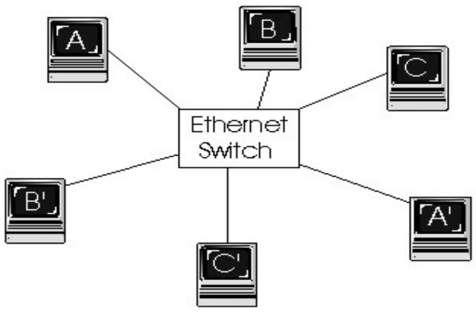


Figure 6-8: An Ethernet switch providing dedicated Ethernet access to six hosts.

One of the advantages of having a switch with a large number of interfaces is that it creates direct connections between hosts and the switch. When a host has a full-duplex direct connection to a switch, it can transmit (and receive) frames at the full transmission rate of its adapter; in particular, the host adapter always senses an idle channel and never experiences a collision. When a host has a direct connection to a switch (rather than a shared LAN connection), the host is said to have **dedicated access**. In Figure 6-8, an Ethernet switch provides dedicated access to six hosts. This dedicated access allows A to send a file to A' while that B is sending a file to B' and C is sending a file to C'. If each host has a 10Mbps adapter card, then the aggregate throughput during the three simultaneous file transfers is 30 Mbps. If A and A' have 100 Mbps adapters and the remaining hosts have 10 Mbps adapters, then the aggregate throughput during the three simultaneous file transfers is 120 Mbps.

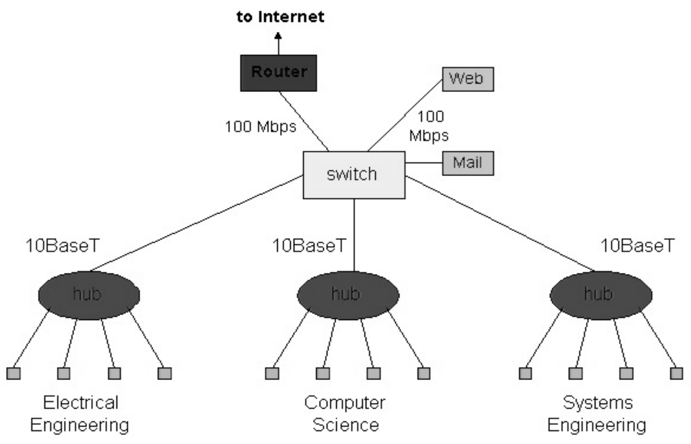


Figure 6-9: An institutional network using a combination of hubs, Ethernet switches and a router.

Figure 6-9 shows how an institution with several departments and several critical servers might deploy a combination of hubs, Ethernet switches and routers. In Figure 6-9, each of the three departments has its own 10 Mbps Ethernet segment with its own hub. Because each departmental hub has a connection to the switch, all intra-departmental traffic is confined to the Ethernet segment of the department (assuming the routing tables in the Ethernet switch are complete). The Web and mail servers each have dedicated 100 Mbps access to the switch. Finally, a router, leading to the Internet, has dedicated 100 Mbps access to the switch. Note that this switch has at least three 10 Mbps interfaces and three100 Mbps interfaces.

6-4-1 Cut-Through Switching

In addition to large numbers of interfaces, support for multitudes of physical media types and transmission rates, and enticing network management features, Ethernet switch manufacturers often tout that their switches use **cut-through switching** rather than store-and-forward packet switching, used by routers and bridges. The difference between store-and-forward and cut-through switching is subtle. To understand this difference considers a packet that is being forwarded through a packet switch (i.e., a router, a bridge, or an Ethernet switch). The packet arrives to the switch on a *inbound link* and leaves the switch on a *outbound link*. When the packet arrives, there may or may not be other packets in the outbound link's output buffer. When there are packets in the output buffer, there is absolutely no difference between store-and-forward and cut-through switching. The two switching techniques only differ when the output buffer is empty.

Recall from unit-1, when a packet is forwarded through a store-and-forward packet switch, the packet is first gathered and stored in its entirety before the switch begins to transmit it on the outbound link. In the case when the output buffer becomes empty before the whole packet has arrived to the switch, this gathering generates a store-and-forward delay at the switch, a delay which contributes to the total end-to-end delay (see Chapter 1). An upper bound on this delay is L/R, where L is the length of the packet and R is transmission rate of the *inbound* link. Note that a packet only incurs a store-and-forward delay if the output buffer becomes empty before the entire packet arrives to the switch.

With cut-through switching, if the buffer becomes empty before the entire packet has arrived, the switch can start to transmit the front of the packet while the back of the packet continues to arrive. Of course, before transmitting the packet on the outbound link, the portion of the packet that contains the destination address must first arrive. (This small delay is inevitable for all types of switching, as the switch must determine the appropriate outbound link.) In summary, with cut-through switching a packet does not have to be fully "stored" before it is forwarded; instead the packet is forwarded through the switch when the output link is free. If the output link is shared with other hosts (e.g., the output link connects to a hub), then the switch must also sense the link as idle before it can "cut-through" a packet.

To shed some insight on the difference between store-and-forward and cut-through switching, let us recall the caravan analogy introduced in unit-I. In this analogy, there is a highway with occasional toll booths, with each toll booth having a single attendant. On the highway there is a caravan of 10 cars traveling together, each at the same constant speed. The cars in the caravan are the only cars on the highway. Each toll booth services the cars at a constant rate, so that when the cars leave the toll booth they are equally spaced apart. As before, we can think of the caravan as being a packet, each car in the caravan as being a bit, and the toll booth service rate as the transmission rate of a link. Consider now what the cars in the caravan do when they arrive to a toll booth. If each car proceeds directly to the toll booth upon arrival, then the toll booth is a "cut-through toll booth". If, on the other hand, each car waits at the entrance until all the remaining cars in the caravan arrive, then the toll booth is "store-and-forward toll booth". The store-and-forward toll booth clearly delays the caravan more than the cut-through toll booth.

A cut-through switch can reduce a packet's end-to-end delay, but by how much? As we mentioned above, the maximum store-and-forward delay is *L/R,* where *L* is the packet size and *R* is the rate of the inbound link. The maximum delay is approximately 1.2 msec for 10 Mbps Ethernet and .12 msec for 100 Mbps Ethernet (corresponding to a maximum size Ethernet packet). Thus, a cut-through switch only reduces the delay by .12 to .2 msec, and this reduction only occurs when the outbound link is lightly loaded. How significant is this delay? Probably not very much in most practical applications, so you may want to think second about selling the family house before investing in the cut-through feature.

	hubs	**bridges**	**routers**	**Ethernet switches**
traffic isolation	*no*	*yes*	*yes*	*yes*
plug and play	*yes*	*yes*	*no*	*yes*
optimal routing	*no*	*no*	*yes*	*no*
cut-through	*yes*	*no*	*no*	*yes*

Figure 6-10: Comparison of the typical features of popular interconnection devices.

We have learned in this section that hubs, bridges, routers and switches can all be used as an interconnection device for hosts and LAN segments. Figure 6-10 provides a summary of the features of each of these interconnection devices. The Cisco Web site provides numerous comparisons of the different interconnection technologies.

7. IEEE 802.11 LANs & PPP: the point-to-point protocol

Structure

Objectives

- Discuss about wireless LAN architecture;
- Discuss format of IEEE 802.11;
- Discuss about point-to-point protocol

7-1 Introduction

In unit-III, we examined the dominant wired LAN protocol - Ethernet. In the previous section we examined how LAN segments can be connected together via hubs, bridges and routers to form larger LANs. In this section we examine a LAN standard (belonging to the same IEEE 802 family as Ethernet) that is being increasingly deployed for un-ether (wireless) LAN communication. The IEEE 802.11 standard, defines the physical layer and media access control (MAC) layer for a wireless local area network. The standard defines three different physical layers for the 802.11 wireless LAN, each operating in a different frequency range and at rates of 1 Mbps and 2 Mbps. In this section we focus on the architecture of 802.11 LANs and their media access protocols. We'll see that although it belongs to the same standard family as Ethernet, it has a significantly different architecture and media access protocol.

7-2 802.11 LAN architecture

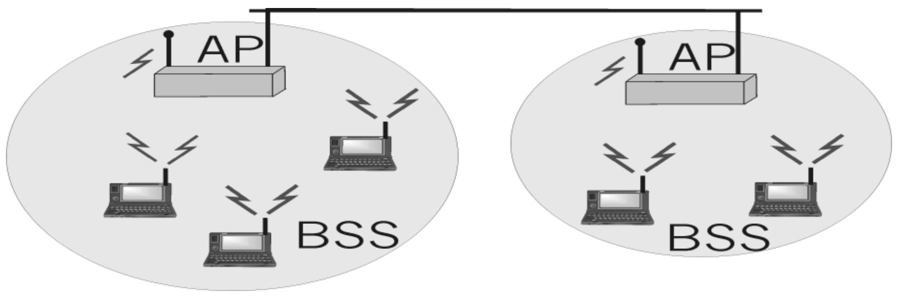


Figure 7-1: IEEE 802.11 LAN architecture

Figure 7-1 illustrates the principal components of 802.11 wireless LAN. The fundamental building block of the 802.11 architecture is the cell, known as the **basic service set (BSS)** in 802.11 parlances. A BSS typically contains one or more wireless stations and a central base station, known as an **access point (AP)** in 802.11 terminologies. The stations, which may be either fixed or mobile, and the central base station communicate amongst themselves using the IEEE 802.11 wireless MAC protocol. Multiple APs may be connected together (e.g., using a wired Ethernet or another wireless channel) to form a so-called **distribution system (DS).** The DS appears to upper level protocols (e.g., IP) as a single 802 network, in much the same way that a bridged, wired 802.3 Ethernet network appears as a single 802 network to the upper layer protocols.

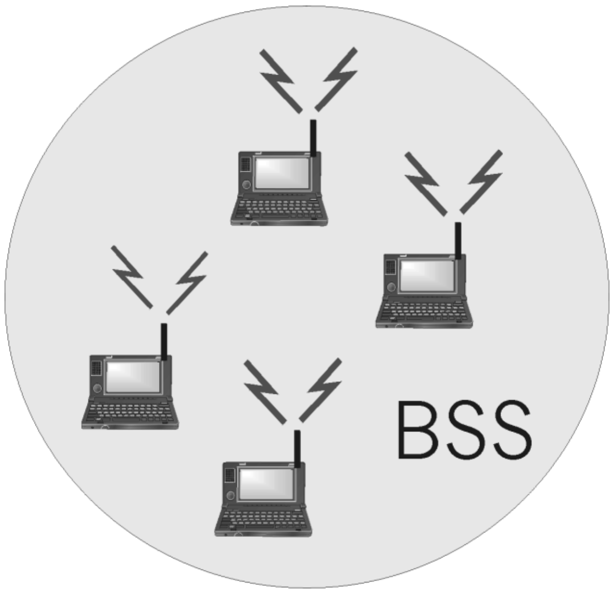


Figure 7-2: An IEEE 802.11 ad hoc network

Figure 7-2 shows that IEEE 802.11 stations can also group themselves together to form an ad hoc network - a network with no central control and with no connections to the "outside world." Here, the network is formed "on the fly," simply because there happen to be mobile devices that have found themselves in proximity to each other, that have a need to communication, and that find no pre-existing network infrastructure (e.g., a pre-existing 802.11 BSS with an AP) in the location. An ad hoc network might be formed, for example, when people with laptops meet together (e.g., in a conference room, a train, or a car) and want to exchange data in the absence of a centralized AP. There has been a tremendous recent increase in interest in ad hoc networking, as communicating portable devices continue to proliferate. Within the IETF, activity in ad hoc networking is centered on the mobile ad hoc networks (manet) working group.

7-3 802.11 Media Access Protocols

Just as in a wired 802.3 Ethernet network, stations in an IEEE 802.11 wireless LAN must coordinate their access and use of the shared communication media (in this case the radio frequency). Once again, this is the job of the media access control (MAC) protocol. The IEEE 802.11 MAC protocol is a carrier sense multiple access protocol with collision avoidance (**CSMA/CA**). Recall from our study of Ethernet in a CSMA protocol first senses the channel to determine if the channel is "busy" with the transmission of a frame from some other station. In the 802.11 specification, the physical layer monitors the energy level on the radio frequency to determine whether or not another station is transmitting and provides this carrier sensing information to the MAC protocol. If the channel is sensed idle for an amount of time equal to or greater than the Distributed Inter Frame Space (DIFS), a station is then allowed to transmit. As with any random access protocol this frame will be successfully received at the destination station if no other station's transmission has interfered with the frame's transmission.

When a receiving station has correctly and completely received a frame for which it was the addressed recipient, it waits a short period of time (known as the Short Inter Frame Spacing - SIFS) and then sends an explicit acknowledgment frame back to the sender. This data link layer acknowledgment lets the sender know that the receiver has indeed correctly received the sender's data frame. We will see shortly that this explicit acknowledgment is needed because, unlike the case of wired Ethernet, a wireless sender can not itself determine whether or not its frame transmission was successfully received at the destination. The transmission of a frame by a sending station and its subsequent acknowledgment by the destination station is shown in Figure 7-3.

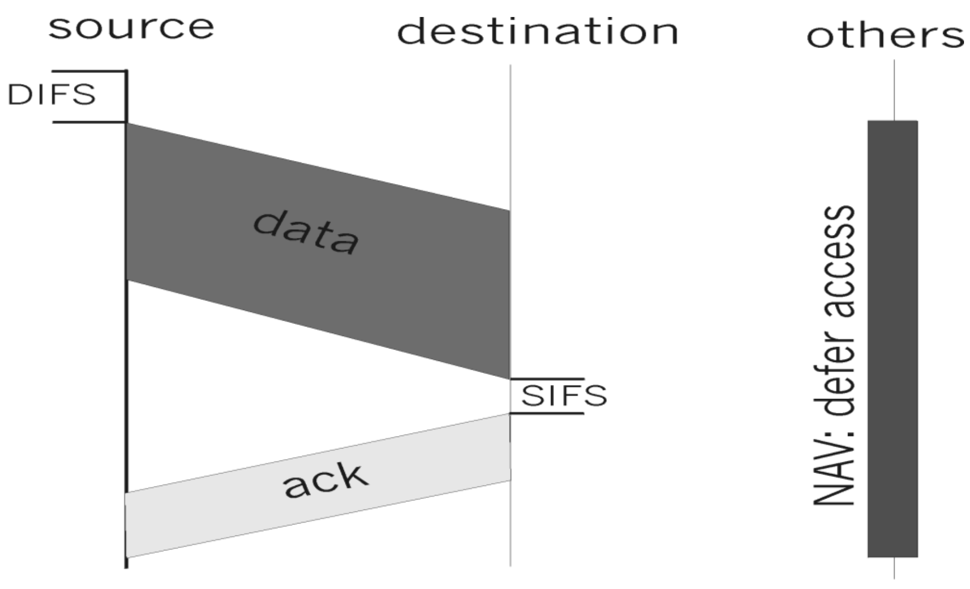


Figure 7-3: data transmission and acknowledgment in IEEE 802.11

Figure 7-3 illustrates the case when the sender senses the channel to be idle. What happens if the sender senses the channel busy? In this case, the station performs a backoff procedure that is similar to that of Ethernet. More specifically, a station that senses the channel busy will defer its access until the channel is later sensed idle. Once the channel is sensed idle for an amount of time equal to DIFS, the station then computes an *additional* random backoff time and counts down this time as the channel is sensed idle. When the random backoff timer reaches zero, the station transmits its frame. As in the case of Ethernet, the random backoff timer serves to avoid having multiple stations immediately begin transmission (and thus collide) after a DIFS idle period. As in the case of Ethernet, the interval over which the backoff timer is randomizes is doubled each time a transmitted frame experiences a collision.

We noted above that unlike the 802.3 Ethernet protocol, the wireless 802.11 MAC protocol does *not* implement collision detection. There are a couple of reasons for this:

- The ability to detect collisions requires the ability to both send (one's own signal) and receive (to determine if another station's transmissions is interfering with one's own transmission) at the same time. This can be costly.
- More importantly, even if one had collision detection and sensed no collision when sending, a collision could still occur at the receiver. This situation results from the particular characteristics of the wireless channel. Suppose that station A is transmitting to station B. Suppose also that station C is transmitting to station B. With the so-called **hidden terminal problem,** physical obstructions in the environment (e.g. a mountain) may prevent A and C from hearing each others transmissions, even though A's and C's transmissions are indeed interfering at the destination, B. This is shown in Figure 7-4(a). A second scenario that results in undetectable collisions at the receiver results from the **fading** of a signal's strength as propagates through the wireless medium. Figure 7-4(b) illustrates the case where A and C are placed such that their signal strengths are not strong enough for them to detect each others' transmissions, and yet their transmissions are strong enough to have interfered with each other at station B.

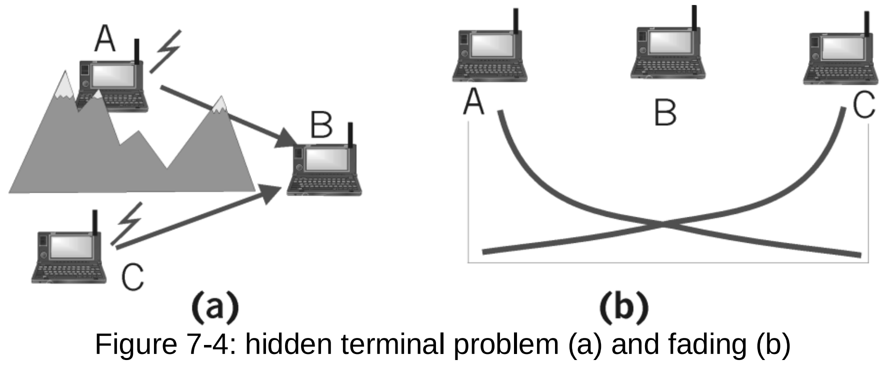


Figure 7-4: hidden terminal problem (a) and fading (b)

Given these difficulties with detecting collisions at a wireless receiver, the designers of IEEE 802.11 developed an access protocol which aimed to avoid collisions (hence the name CSMA/CA), rather than detect and recover from collisions (CSMA/CD). First, the IEEE 802.11 frame contains a duration field in which the sending station explicit indicates the length of time that its frame will be transmitting on the channel. This value allows other stations to determine the minimum amount of time (the so-called network allocation vector, NAV) for which they should defer their access, as shown in Figure 7.3.

The IEEE 802.11 protocol can also use a short Request to Send (RTS) control frame and a short Clear to Send (CTS) frame to *reserve* access to the channel. When a sender wants to send a frame, it can first send a RTS frame to the receiver, indicating the duration of the data packet and the ACK packet. A receiver that receives an RTS frame responds with a CTS frame, giving the sender explicit permission to send. All other stations hearing the RTS or CTS then know about the pending data transmission and can avoid interfering with those transmissions. The RTS, CTS, DATA and ACK frames are shown in Figure 7-5. An IEEE 802.11 sender can operate either using the RTS/CTS control frames, as shown in Figure 7-5, or can simply send its data without first using the RTS control frame, as shown in Figure 7-3.

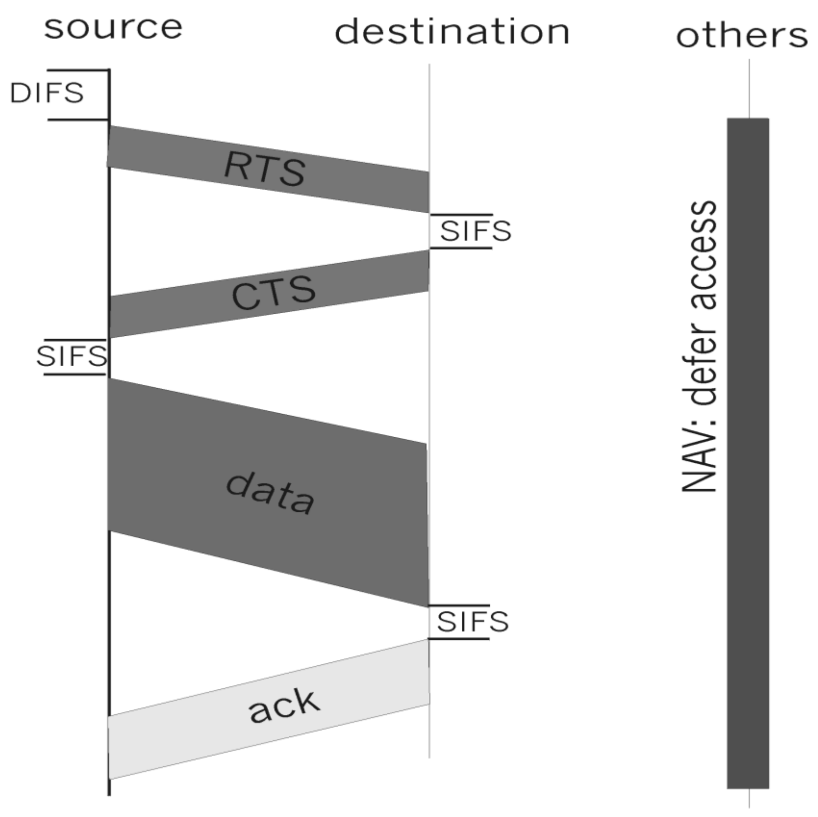


Figure 7-5: Collision Avoidance using the RTS and CTS frames

The use of the RTS and CTS frames helps avoid collisions in three important ways:

- Because the receiver's transmitted CTS frame will be heard by all stations within the receiver's vicinity, the CTS frame helps avoid both the hidden station problem and the fading problem.
- Because the RTS and CTS frames are short, a collision involving a RTS or CTS frame will only last for the duration of the whole RTS or CTS frame. Note that when the RTS and CTS frames are correctly transmitted, there should be no collisions involving the subsequent DATA and ACK frames.

In our discussion above, we have only highlighted some of the key aspects of the 802.11 protocol. Additional protocol capabilities such as time synchronization, power management, joining and leaving a network (i.e., roaming stations) are covered in the full IEEE 802.11 standard.

7-4 PPP: the point-to-point protocol

Most of our discussion of data link protocols thus far has focused on protocols for broadcast channels. In this section we cover a data link protocol for point-to-point links - PPP, the Point-to-Point protocol. Because PPP is typically the protocol of choice for a dialup link from residential hosts, it is undoubtedly one of the most widely-deployed data link protocols today. The other important data link protocol in use today is the HDLC (High Level Data Link Control) protocol. Our discussion here of the simpler PPP protocol will allow us to explore many of the most important features of point-to-point data link protocol.

As its name implies, the Point-to-Point Protocol (PPP) is a data link layer protocol that operates over a **point-to-point link** - a link connecting two communicating link-level peers, one on each end of the link The point-to-point link over which PPP operates might be a serial dialup telephone line (e.g., a 56K modem connection), a SONET/SDH link, an X.25 connection, or over an ISDN circuit. An noted above, PPP has become the protocol of choice for connecting home users to their ISP's over a dialup connection.

Before diving into the details of PPP, it is instructive to examine the original requirements that the IETF placed on the design of:

- *Packet framing.* The PPP protocol data link layer sender must be able to take a network-level packet and frame (encapsulate) it within the PPP data link layer frame such that the receiver will be able to identify the start and end of both the data link frame, and the network layer packet within the frame.
- *Transparency.* The PPP protocol must not place any constraints on data appearing on the network layer packet (headers or data).

Thus, for example, the PPP protocol can not forbid the use of certain bit patterns in the network layer packet. We'll return this issue shortly in our discussion of byte stuffing below.

- *Multiple network layer protocols.* The PPP protocol must be able to support multiple network layer protocols (e.g., IP and DECnet) running over the *same* physical link at the *same* time. Just as the IP protocol is required to multiplex different transport level protocols (e.g., TCP and UDP) over a single end-to-end connection, so too must PPP be able to multiplex different network layer protocols over a single point-to-point connection. This requirement means that at a minimum, PPP will likely require a "protocol type" field or some similar mechanism so the receiving side PPP can demultiplex a received frame up to the appropriate network layer protocol.
- *Multiple types of links.* In addition to being able to carry multiple higher level protocols, PPP must also be able to operate over a wide variety of link types, including links that are either serial (transmitting a bit at a time in a given direction) or parallel (transmitting bits in parallel), synchronous (transmitting a clock signal along with the data bits) or asynchronous, low speed or high speed, electrical or optical.
- *Error detection.* A PPP receiver must be able to detect bit errors in the received frame.
- *Connection liveness.* PPP must be able to detect a failure at the link level (e.g., the inability to transfer data from the sending side of the link to the receiving side of this link) and signal this error condition to the network layer.
- *Network Layer Address Negotiation.* PPP must provide a mechanism for the communicating network layers (e.g., IP) to learn or configure each other's network layer address.
- *Simplicity.* PPP was required to meet a number of additional requirements beyond the seven listed above. On top of all of these requirements, first and foremost among all of the PPP requirements is that of "simplicity." RFC 1547 states "the watchword for a point-to-point protocol should be simplicity." A tall order indeed given all of the other requirements placed on the design of PPP! More than 50 RFC's now define the various aspects of this "simple" protocol.

While it may appear that many requirements were placed on the design of PPP, the situation could actually have been much more difficult! The design specifications for PPP also explicitly note protocol functionality that was PPP was *not* required to implement:

- *Error correction.* PPP is required to detect bit errors but is *not* required to correct them.
- *Flow control.* A PPP receiver is expected to be able to receive frames at the full rate of the underlying physical layer. If a higher layer can not receive packets at this full rate, it is then up to the higher layer to drop packets or throttle the sender at the higher

layer. That is, rather than having the PPP sender throttle its own transmission rate, it is the responsibility of a higher level protocol to throttle the rate at which packets are delivered to PPP for sending.

- *Sequencing.* PPP is *not* required to deliver frames to the link receiver in the same order in which they were sent by the link sender. It is interesting to note that while this flexibility is compatible with the IP service model (which allows IP packets to be delivered end-to-end in any order), other network layer protocols which operate over PPP do require sequenced end-to-end packet delivery.
- *Multipoint links.* PPP need only operate over links that have a single sender and a single receiver. Other link layer protocols (e.g., HDLC) can accommodate multiple receivers (e.g., an Ethernet-like scenario) on a link.

Having now considered the design goals) and non-goals) for PPP, let us see how the design of PPP met these goals.

7-5 PPP Data Framing

Figure 7-6 shows a PPP data frame using HDLC-like framing.

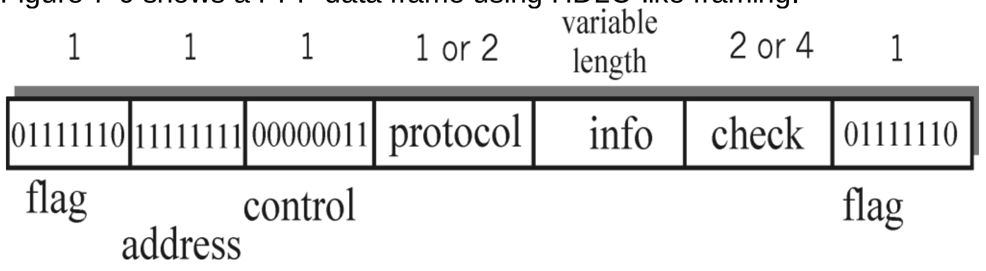


Figure 7-6: PPP data frame format

The PPP frame contains the following fields:

- *Flag field.* Every PPP frame begins and ends with a 1 byte flag field with a value of 01111110.
- Address field. The only possible value for this field is 11111111.
- *Control Field.* The only possible value of this field is 00000011. Because both the address and control fields can currently take only a fixed value, one wonders why the fields are even defined in the first place. The PPP specification [RFC 1622] states that other values "may be defined at a later time," although none have been defined to date. Because these fields take fixed values, PPP allows the sender to simply not send the address and control bytes, thus saving two bytes of overhead in the PPP frame.
- *Protocol.* The protocol field tells the PPP receiver the upper layer protocol to which the received encapsulated data (i.e., the contents of the PPP frame's info field) belongs. On receipt of a PPP frame, the PPP receiver will check the frame for correctness and then pass the encapsulated data on to the appropriate protocol. [RFC 1700]

defines the 16-bit protocol codes used by PPP. Of interest to us are the IP protocol (i.e., the data encapsulated in the PPP frame is an IP datagram) which has a value of 21 hexadecimal, other network layer protocols such as Appletalk (29) and DECnet (27), the PPP link control protocol (c021 hexadecimal) that we discuss in detail in the following section, and the IP Control Protocol (8021) which is called by PPP when a link is first activated in order to configure the IP-level connection between the two routers on each end of the link (see below).

- *Information.* This field contains the encapsulate packet (data) that is being sent by an upper layer protocol (e.g., IP) over the PPP link. The default maximum length of the information field is 1500 bytes, although this can be changed when the link is first configured, as discussed below.
- *Checksum.* The checksum field is used to detect bit errors in a transmitted frame. It uses either a two or four byte HDLC-standard cyclic redundancy code.

7-5-1 Byte Stuffing

Before closing our discussion of PPP framing, let us consider a problem that arises when any protocol uses a specific bit pattern (flag field) to delineate the beginning or end of the frame: what happens if the flag pattern itself occurs elsewhere in the packet? For example, what happens if the flag field value of 01111110 appears in the information field? Will the receiver incorrectly detect the end of the PPP frame?

One way to solve this problem would be for PPP to forbid the upper layer protocol from sending data containing the flag field bit pattern. The PPP requirement of transparency discussed above obviates this possibility. An alternate solution, and the one taken in PPP and many other protocols, is to use a technique known as **byte stuffing**.

PPP defines a special control escape byte, 01111101. If the flag sequence, 01111110 appears anywhere in the frame, except in the flag field, PPP precedes that instance of the flag pattern with the control escape byte. That is, it "stuffs" (adds) a control escape byte into the transmitted data stream, before the 01111110, to indicate that the following 011111110 is *not* a flag value but is, in fact, actual data. A receiver that sees a 01111110 proceeded by a 01111101 will, of course, remove the stuffed control escape to reconstruct the original data. Similarly, if the control escape byte bit pattern itself appears as actual data, it too must be preceded by a stuffed control escape byte. Thus, when the receiver sees a single control escape byte by itself in the data stream, it knows that the byte was stuffed into the data stream. A pair of control escape bytes occurring back-to-back means that one instance of the control escape byte appears in the original data being sent. Figure 7-7 illustrates PPP byte stuffing. (Actually, PPP also XORs the data byte being escaped with 20 hexadecimal, a detail we omit here for simplicity).

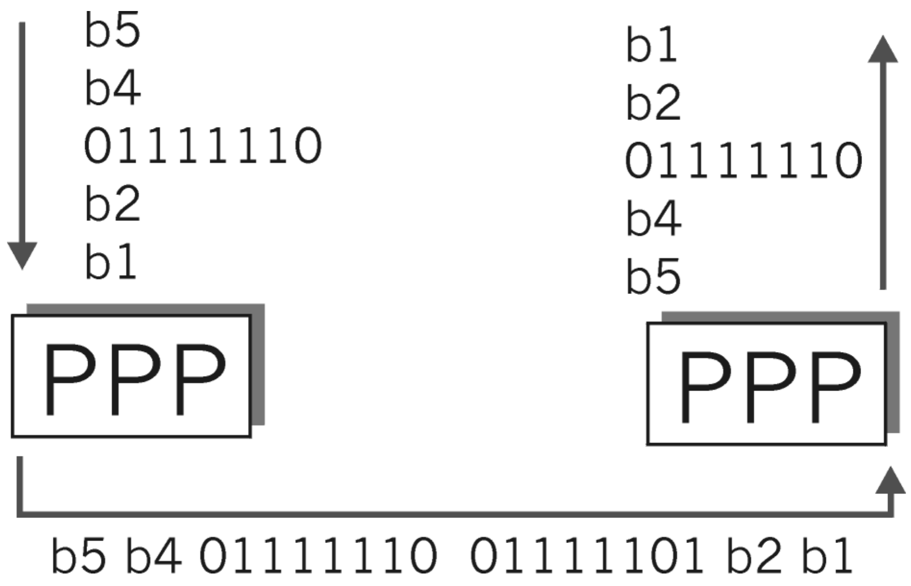


Figure 5.8-2: byte stuffing

7-7 PPP Link Control Protocol (LCP) and network control protocols

Thus far, we have seen how PPP frames the data being sent over the point-to-point link. But how does the link get initialized when a host or router on one end of the PPP link is first turned on? The initialization, maintenance, error reporting, and shutdown of a PPP link is accomplished using PPP's Link Control Protocol (LCP) and family of PPP network control protocols.

Before any data is exchanged over a PPP link, the two peers (one at each end of the PPP link) must first perform a considerable amount of work to configure the link, in much the same way that a TCP sender and receiver must perform a three-way handshake to set the parameters of the TCP connection before TCP data segments are transmitted. Figure 7-8 illustrates the state transition diagram for the LCP protocol for configuring, maintaining and terminating the PPP link.

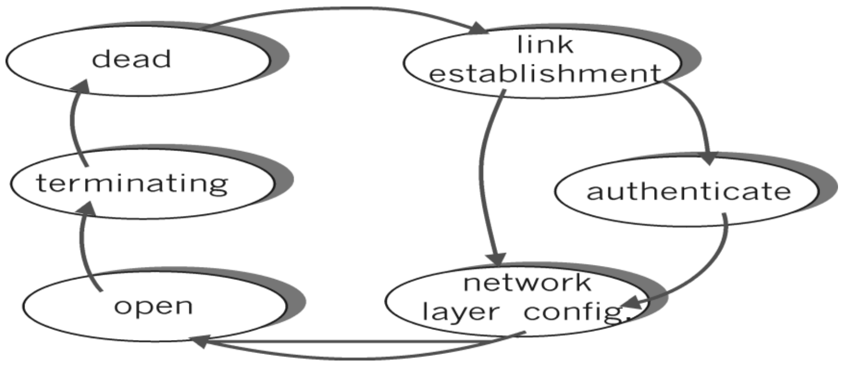


Figure 7-8: PPP Link Control Protocol

The PPP link always begins and ends in the dead state. When an event such as a carrier detection or network administrator intervention indicates that a physical layer is present and ready to be used, PPP enters the link establishment state. In this state, one end of the link sends its desired link configuration options using an LCP configure-request frame (a PPP frame with the protocol field set to LCP and the PPP information field containing the specific configuration request). The other side then responds with a configure-ack frame (all options acceptable), a configure-nak frame (all options understood but not acceptable) or a configure-reject frame (options not recognizable or not acceptable for negotiation). LCP configuration options include a maximum frame size for the link, the specification of an authentication protocol (if any) to be used, and an option to skip the use of the address and control fields in the PPP frames.

Once the link has been established, link options negotiated, and the authentication (if any) performed, the two sides of the PPP link then exchange network-layer-specific network control packets with each other. If IP is running over the PPP link, the IP Control Protocol [RFC 1332] is used to configure the IP protocol modules at each end of the PPP link. IPCP packets are carried within a PPP frame (with a protocol field value of 8021), just as LCP packets are carried in a PPP frame. IPCP allows the two IP modules to exchange or configure their IP addresses and negotiate whether or not IP packets will be set in compressed form. Similar network control protocols are defined for other network layer protocols, such as DECnet [RFC 1762] and AppleTalk [RFC 1378]. Once the network layer has been configured, PPP may then begin sending network-layer datagrams - the link is in the opened state and data has begun to flow across the PPP link. The LCP echo-request packet and echo-reply packet can be exchanged between the two PPP endpoints in order to check the status of the link.

The PPP link remains configured for communication until an LCP terminate-request packet is sent. If a terminate-request LCP packet is sent by one end of the PPP link and replied to with a terminate-ack LCP packet, the link then enters the dead state.

In summary, PPP is a data link layer protocol by which two communicating link-level peers, one on each end of a point-to-point link, exchange PPP frames containing network layer datagrams. The principal components of PPP are:

- *Framing.* A method for encapsulating data in a PPP frame, identifying the beginning and end of the frame, and detecting errors in the frame.
- *Link Control Protocol.* A protocol for initializing, maintaining, and taking down the PPP link.
- *Network control protocols.* A family of protocols, one for each upper layer network protocol, that allows the network layer modules to

configure themselves before network-level datagrams begin flowing across the PPP link.

Summary

In this unit, we've examined the data link layer - its services, the principles underlying its operation, and a number of important specific protocols that use these principles in implementing data link services.

We saw that the basic service of the data link layer is to move a network-layer datagram from one node (router or host) to an adjacent node. We saw that all data link protocols operate by encapsulating a network-layer datagram within a link-layer frame before transmitting the frame over the "link" to the adjacent node. Beyond this common framing function, however, we learned that different data link protocols can provide very different link access, delivery (reliability, error detection/correction), flow control, and transmission (e.g., full-duplex versus half-duplex) services. These differences are due in part to the wide variety of link types over which data link protocols must operate. A simple point-to-point link has a single sender and receiver communicating over a single "wire." A multiple access link is shared among many senders and receivers; consequently the data link protocol for a multiple access channel has a protocol (its multiple access protocol) for coordinating link access. In the cases of ATM, X.25 and frame relay, we saw that the "link" connecting two adjacent nodes (e.g., two IP routers that are adjacent in an IP sense - that they are next-hop IP routers towards some destination), may actually be a *network* in and of itself. In one sense, the idea of a network being considered as a "link" should not seem odd. A telephone "link" connecting a home modem/computer to a remote modem/router, for example, is actually a path through a sophisticated and complex telephone *network*.

Among the principles underlying data link communication, we examined error detection and correction techniques, multiple access protocols, link-layer addressing, and the construction of extended local area networks via hubs. Bridges and switches. In the case of error detection/correction, we examined how it is possible to add additional bits to a frame's header that are used to detect, and in some cases correct, bit-flip errors that might occur when the frame is transmitted over the link. We covered simple parity and check-summing schemes, as well as the more robust cyclic redundancy check. We then moved on to the topic of multiple access protocols. We identified and studied three broad approaches for coordinating access to a broadcast channel: channel partitioning approaches (TDM, FDM, CDMA), random access approaches (the ALOHA protocols, and CSMA protocols), and taking-turns approaches (polling and token passing). We saw that a consequence of having multiple nodes share a single broadcast channel was the need to provide node address at the data link level. We learned that physical addresses were quite different from network-layer addresses, and that in the case of the Internet, a special protocol (ARP - the address resolution protocol) is used to translate between these two forms of addressing. We then examined how nodes sharing a broadcast channel form a local area network (LAN), and how multiple LANs can be connected together to form larger LANs - all *without* the intervention of network-layer routing to interconnect these local nodes. Finally, we covered a number of specific data link layer protocols in detail - Ethernet, the wireless IEEE 802.11 protocol, and the Point-to-Point protocol, PPP. As discussed in sections 5.9 and 5.10, ATM, X.25, and frame relay can also be used to connect two network-layer routers. For example, in the IP-over-ATM scenario, two adjacent IP routers can be connected to each other by a virtual circuit through an ATM network. In such circumstances, a network that is based on one network architecture (e.g., ATM, or frame relay) can serve as a single logical link between two neighboring nodes (e.g., IP routers) in another network architecture..

Having covered the data link layer, *our journey down the protocol stack is now over*! Certainly, the physical layer lies below the data link layer, but the details of physical layer is the topic probably best left for another course (e.g., in communication theory, rather than computer networking). We have, however, touched upon several aspects of the physical layer in this unit (e.g., our brief discussions of Manchester encoding and of signal fading) and in unit 1.

Although our journey down the protocol stack is over, our study of computer networking is not yet over. In the following three chapters we cover multimedia networking, network security, and network management. These three topics do not fit conveniently into any one layer; indeed, each topic crosscuts many layers. Understanding these topics (sometimes billed as "advanced topics" in some networking texts) thus requires a firm foundation in all layers of the protocol stack - a foundation that is now complete with our completed study of the data link layer!

Exercise

1) If all the links in the Internet were to provide the reliable-delivery service, would the TCP reliable-delivery service be completely redundant? Why or why not?

2) What are some of possible services that a link-layer protocol can offer to the network layer? Which of these link-layer services have corresponding services in IP? And In TCP?

3) Suppose the information content of a packet is the bit pattern 1010101010101011 and an even parity scheme is being used. What would be the value of the checksum field in a single parity scheme?

4) Suppose two nodes start to transmit at the same time a packet of length L over a broadcast channel of rate R. Denote the propagation delay between the two nodes as t_{prop}. Will there be a collision if $t_{prop} < L/R$? Why or why not?

5) What are human cocktail analogies for polling, and token passing protocols?

6) Why would the token-ring protocol be inefficient if the LAN has a very large perimeter?

7) How big is the LAN address space? The IPv4 address space? The IPv6 address space?

8) Suppose nodes A, B, and C each attach to the same broadcast LAN (through their adapters). If A sends thousands of frames to B with each frame addressed to the LAN address of B, will C's adapter process these frames? If so, will C's adapter pass the IP datagram's in these frames to C (i.e., the adapter's parent node)? How will your answers change if A sends frames with the LAN broadcast address?

9) Why is an ARP query sent within a broadcast frame? Why is an ARP response sent within a frame with a specific LAN address?

10) Compare the frame structures for 10BaseT, 100BaseT and Gigabit Ethernet. How do they differ?

11) Suppose a 10 Mbps adapter sends into a channel an infinite stream of 1s using Manchester encoding. The signal emerging from the adapter will have how many transitions per second?

12) After the 5th collision, what is the probability that the value of K that a node chooses is 4? The result K=4 corresponds to a delay of how many seconds on a 10 Mbps Ethernet.

13) Does the TC sub layer at the transmitter fill in any of the fields in the ATM header? Which ones

14) In the IEEE 802.11 specification, the length of the SIFS period must be shorter than the DIFS period. Why?

15) Suppose the IEEE 802.11 RTS and CTS frames were as long as the standard DATA and ACK frames. Would there be any advantage to using the CTS and RTS frames? Why?

GLOSAARY

100Base-FX: A two-wire fiber implementation of Fast Ethernet.

100Base-T4: A four-wire UTP implementation of Fast Ethernet.

100Base-TX: A two-wire UTP implementation of Fast Ethernet.

10Base2: The thin coaxial cable implementation of Standard Ethernet.

10Base5: The thick coaxial cable implementation of Standard Ethernet.

10Base-F: The fiber implementation of Standard Ethernet.

10Base-T: The twisted-pair implementation of Standard Ethernet.

10Base-E: The extended implementation of Ten-Gigabit Ethernet.

10Base-L: A fiber implementation of Ten-Gigabit Ethernet using long-wave laser.

10Base-S: A fiber implementation of Ten-Gigabit Ethernet using short-wave laser.

1-persistent strategy: A CSMA persistence strategy in which a station sends frames immediately if the line is idle.

56K modems: A modems technology using two different data rates: one for uploading and one for downloading from the Internet.

Abstract Syntax Notation 1 (ASN.1): A standard for representing simple and structured data.

Access control: The determination of link control through a data link protocol.

Access rate: In Frame Relay, the data rate that can never be exceeded.

Access point: A central base station in a BSS.

Acknowledgment (ACK): A response sent by the receiver to indicate the successful receipt and acceptance of data.

Active close: In the client-server model, the closing of a communication by the client.

Active document: In the World Wide Web, a document executed at the local site using Java.

Active open: In the client-server model, the opening of a communication by the client.

Adaptive delta modulation: A delta modulation technique in which the value of delta changes according to the amplitude of the analog signal.

Additive increase: With slow start, a congestion avoidance strategy in which the window size is increased by just one segment instead of exponentially.

Address Resolution Protocol (ARP): In TCPIP, a protocol for obtaining the physical address of a node when the Internet address is known.

Advanced Encryption Standard (AES): A secret-key cryptosystem adapted by NIST it replace DES.

Address aggregation: A mechanism in which the blocks of addresses for several organizations are aggregated into one larger block.

Address space The total number of addresses used by a protocol.

Address-mask request and reply ICMP: Messages that find the network mask.

Advanced Mobile Phone System (AMPS): A North American analog cellular phone system using FDMA.

Advanced Research Projects Agency (ARPA): The government agency that funded ARPANET.

Advanced Research Projects Agency Network (ARPANET): The packet-switching network that was funded by ARPA.

ALOHA: The original random multiple access method in which a station can send a frame any time it has one to send.

American National Standards Institute (ANSI): A national standards organization that defines standards in the United States.

American Standard Code for Information Interchange (ASCII): A character code developed by ANSI and used extensively for data communication.

Amplitude: The strength of a signal, usually measured in volts, amperes, or watts.

Analog: A continuously varying entity.

Analog data: Data that is continuous and smooth and not limited to a specific number of values.

Analog hierarchy: A telephone company system in which multiplexed signals is combined into successively larger groups for more efficient transmission.

Analog leased service: A service featuring a dedicated line between two users.
Analog signal: A continuous waveform that changes smoothly over time.
Analog switched service: A temporary analog connection between two users.
Analog-to-analog modulation: The representation of analog information by an analog signal.

Analog-to-digital conversion: The representation of analog information by a digital signal.

Angle of incidence: In optics, the angle formed by a light ray approaching the interface between two media and the line perpendicular to the interface.

Aperiodic signal: A signal that does not exhibit a pattern or repeating cycle.
Applet: A computer program for creating an active Web document. It is usually written in Java.

Application adaptation layer (AAL): A layer in ATM protocol that breaks user data into 48-byte payloads.

Application layer: The fifth layer in the Internet model; provides access to network resources.

Application programming interface (API): A set of declarations, definitions, and procedures followed by programmers to write client-server programs.
Area A collection of networks, hosts, and routers all contained within an autonomous system.

Area border router A router inside an area that summarizes the information about the area and sends it to other areas.
Area identification: A 32-bit field that defines the area within which the routing takes place.

Asymmetric digital subscriber line (ADSL): A communication technology in which the downstream data rate is higher than the upstream rate.
Asynchronous balanced mode (ABM): In HDLC, a communication mode in which all stations are equal.

Asynchronous connectionless link (ACL) : A link between a Bluetooth master and slave in which a corrupted payload is retransmitted.

Asynchronous Transfer Mode (ATM): A wide area protocol featuring high data rates and equal-sized packets (cells); ATM is suitable for transferring text, audio, and video data.

Asynchronous transmission: Transfer of data with start and stop bit(s) and a variable time interval between data units.

ATM LAN: A LAN using ATM technology.

ATM layer: A layer in ATM that provides routing, traffic management, switching, and multiplexing services.

ATM switch: An ATM device providing both switching and multiplexing functions.
Attachment unit interface (AUI): A 10Base5 cable that performs the physical interface functions between the station and the transceiver.
Attenuation: The loss of a signal's energy due to the resistance of the medium.
Audio: Recording or transmitting of sound or music.

Authenticating state: In PPP, an optional state that verifies the identity of the receiver.

Authentication Header (AH) Protocol: A protocol defined by IPSec at the network layer that provides integrity to a message through the creation of a digital signature by a hashing function.

Authentication server (AS): The KDC in the Kerberos protocol. **Authentication:** Verification of the sender of a message.

Automatic repeat request (ARQ): An error-control method in which correction is made by retransmission of data.

Automatic tunneling: Tunneling in which the receiving host has an IPv6 compatible address; no reconfiguration is necessary.

Autonegotiation: A Fast Ethernet feature that allows two devices to negotiate the mode or data rate.

Autonomous system (AS): A group of networks and routers under the authority of a single administration.

Autonomous system boundary router: Routers responsible for dissipating information about other autonomous systems into the current system.
Available bit rate (ABR): The minimum data rate in ATM at which cells can be delivered.

Back off: In multiple access, waiting before re-sending after a collision. **Backbone router:** A router inside the backbone.

Backbone A network that connects smaller networks in an organization. **Backward explicit congestion notification (BECN)** A bit in the Frame Relay packet that notifies the sender of congestion.

Bandwidth on demand A digital service that allows subscribers higher speeds through the use of multiple lines.

Bandwidth The difference between the highest and the lowest frequencies of a composite signal. It also measures the information-carrying capacity of a line or a network.

Bandwidth-delay product A measure of the number of bits that can be sent while waiting for news from the receiver.

Banyan switch A multistage switch with microswitches at each stage that route the packets based on the output port represented as a binary string. **Barker sequence** A sequence of 11 bits used for spreading.

Baseband transmission Transmission of digital or analog signal without modulation using a low-pass channel.

Band-pass channel A channel that can pass a range of frequencies.

Base header In IPv6, the main header of the datagram.

Baseline wandering In decoding a digital signal, the receiver calculates a running average of the received signal power. This average is called the baseline. A long string of 0s and 1s can cause a drift in the baseline (baseline wandering) and make it difficult for the receiver to decode correctly.
Basic Encoding Rule (BER) A standard that encodes data to be transferred through a network.

Basic Latin ASCII character set.

Basic service set (BSS) The building block of a wireless LAN as defined by the IEEE 802.11 standard.

Baud rate The number of signal elements transmitted per second. A signal element consists of one or more bits.

Bayone-Neill-Concelman (BNC) connector A common coacxial cable connector.

Best-effort delivery The unreliable transmission mechanism by IP that does not guarantee message delivery.

Bidirectional authentication An authentication method involving a challenge and a response from sender to receiver and vice versa.

Bidirectional frame (B-frame) An MPEG frame that is related to the preceding and following I-frame or P-frame.

Binary exponential backup In contention access methods, a retransmission delay strategy used by a system to delay access.

Binary notation Representation of IP addresses in binary.

Biphase A type of polar encoding where the signal changes at the middle of the bit interval. Manchester and differential Manchester are examples of biphase encoding.

Bipolar encoding A digital-to-digital encoding method in which 0 amplitude represents binary 0 and positive and negative amplitudes represent alternate 1s.

Bipolar with 8-zero substitution (B8ZS) A scrambling technique in which a stream of 8 zeros are replaced by a predefined pattern to improve bit synchronization.

Bipolar n-zero substitution (BnZS) An encoding method to provide synchronization for long strings of 0s.

Bit binary digit; the smallest unit of data(0 or 1)

Bit interval The time required to send one bit.

Bit padding In TDM, the addition of extra bits to a device's source stream to force speed relationships.

Bit rate The number of bits transmitted per second.

Bit stuffing In a bit-oriented protocol, the process of adding an extra bit in the data section of a frame to prevent a sequence of bits from looking like a flag.

Bit-oriented protocol A protocol in which the data frame is interpreted as a sequence of bits.

Bits per second (bps) A measurement of data speed; bits transmitted per second.

Block cipher An encryptiondecryption algorithm that has a block of bits as its basic unit.

Block code An error detectioncorrection code in which data are divided into units called datawords. Redundant bits are added to each dataword to create a codeword.

Block coding A coding method to ensure synchronization and detection of errors.
Blocking An event that occurs when a switching network is working at its full capacity and cannot accept more input.

Blocking port A port on a bridge that does not forward a frame.
Bluetooth A wireless LAN technology designed to connect devices of different functions such as telephones and notebooks in a small area such as a room.

BNC connector A common coaxial cable connector.

Bootstrap Protocol (BOOTP) The protocol that provides configuration information from a table (file).

Border Gateway Protocol (BGP) An interautonomous system routing protocol based on path vector routing.

Bridge A network device operating at the first two layers of the Internet model with filtering and forwarding capabilities.

Broadband transmission Transmission of signals using modulation of a higher frequency signal. The term implies a wide-bandwidth data combined from different sources.

Broadcast address An address that allows transmission of a message to all nodes of a network.

Broadcastunknown server (BUS) A server connected to an ATM switch that can multicast and broadcast frames.

Broadcasting Transmission of a message to all nodes in a network.
Browser An application program that displays a WWW document. A browser usually uses other Internet services to access the document.

BSS-transition mobility In a wireless LAN, a station that can move from one BSS to another but is confined inside one ESS.

Bucket brigade attack See *man-in-the middle attack*

Burst error Error in a data unit in which two or more bits have been altered.
Bursty data Data with varying instantaneous transmission rates.

Bus topology A network topology in which all computers are attached to a shared medium.

Byte A group of eight bits.

Byte stuffing In a byte-oriented protocol, the process of adding an extra byte in the data section of a frame to prevent a byte from looking like a flag.
Byte-oriented protocol A protocol in which the data section of the frame is interpreted as a sequence of bytes (characters).

Cable modem A technology in which the TV cable provides Internet access.
Cable modem transmission system (CMTS) A device installed inside the distribution hub that receives data from the Internet and passes them to the

combiner.

Cable TV network A system using coaxial or fiber optic cable that brings multiple channels of video programs into homes.

Caching The storing of information in a small, fast memory.

Caesar cipher A shift cipher used by Julius Caesar with the key value of 3.

Carrier extension A technique in Gigabit Ethernet that increases the minimum length of the frame to achieve a higher maximum cable length.

Carrier sense multiple access (CSMA) A contention access method in which each station listens to the line before transmitting data.

Carrier sense multiple access with collision avoidance (CSMACA) An access method in which collision is avoided.

Carrier sense multiple access with collision detection (CSMACD) An access method in which stations transmit whenever the transmission medium is available and retransmit when collision occurs.

Carrier signal A high frequency signal used for digital-to-analog or analog-to-analog modulation. One of the characteristics of the carrier signal (amplitude, frequency, or phase) is changed according to the modulating data.

Cell A small, fixed-size data unit; also, in cellular telephony, a geographical area served by a cell office.

Cell network A network using the cell as its basic data unit.

Cellular telephony A wireless communication technique in which an area is divided into cells. A cell is served by a transmitter.

Certification Authority (CA) An agency such as a federal or state organization that binds a public key to an entity and issues a certificate.

Challenge Handshake Authentication Protocol (CHAP) In PPP, a three-way handshaking protocol used for authentication.

Channel A communications pathway.

Channelization A multiple access method in which the available bandwidth of a link is shared in time.

Character-oriented protocol See *byte-oriented protocol*

Checksum A field used for error detection. It is formed by adding bit streams using one's complement arithmetic and then complementing the result.

Chip In CDMA, a number in a code that is assigned to a station.

Choke point A packet sent by a router to the source to inform it of congestion.

Chunk A unit of transmission in SCTP.

Cipher An encryptiondecryption algorithm.

Cipher block chaining (CBC) mode A DES and triple DES operation mode in which the encryption (or decryption) of a block depends on all previous blocks.

Cipher feedback mode (CFM) A DES and triple DES operation mode in which data is sent and received 1 bit at a time, with each bit independent of the previous bits.

Cipher stream mode (CSM) A DES and triple DES operation mode in which data is sent and received 1 byte at a time.

Cipher suite AA list of possible ciphers.

Ciphertext The encrypted data.

Circuit switching A switching technology that establishes an electrical connection between stations using a dedicated path.

Cladding Glass or plastic surrounding the core of an optical fiber; the optical density of the cladding must be less than that of the core.
Class A address An IPv4 address with the first octet between 0 and 127.
Class B address An IPv4 address with the first octet between 128 and 191.
Class C address An IPv4 address with the first octet between 192 and 223.
Class D address An IPv4 multicast address.

Class E address An IPv4 address reserved for special purposes.

Classful addressing An IPv4 addressing mechanism in which the IP address space is divided into 5 classes: A, B, C, D, and E. Each class occupies some part of the whole address space.

Classless addressing An addressing mechanism in which the IP address space is not divided into classes.

Classless InterDomain Routing (CIDR) A technique to reduce the number of routing table entries when supernetting is used.

Client process A running application program on a local site that requests service from a running application program on a remote site.

Client-server model The model of interaction between two application programs in which a program at one end (client) requests a service from a program at the other end (server).

Closed-loop congestion control A method to alleviate congestion after it happens.
Coaxial cable A transmission medium consisting of a conducting core, insulating material, and a second conducting sheath.

Code division multiple access (CDMA) A multiple access method in which one channel carries all transmissions simultaneously.

Codeword The encoded dataword.

ColdFusion A dynamic web technology that allows the fusion of data items coming from a conventional database.

Collision The event that occurs when two transmitters send at the same time on a channel designed for only one transmission at a time; data will be destroyed.
Collision domain The length of the medium subject to collision.

Committed burst size The maximum number of bits in a specific time period that a Frame Relay network must transfer without discarding any frames.
Committed information rate (CIR) The committed burst size divided by time.
Common carrier A transmission facility available to the public and subject to public utility regulation.

Common Gateway Interface (CGI) A standard for communication between HTTP servers and executable programs. CGI is used in creating dynamic documents.

Community antenna TV (CATV) A cable network service that broadcasts video signals to locations with poor or no reception.

Compatible address An IPv6 address consisting of 96 bits of zero followed by 32 bits of IPv4.

Competitive local exchange carrier (CLEC) A telephone company that cannot provide main telephone services; instead, other services such as mobile telephone service and toll calls inside a LATA are provided.
Complementary code keying (CCK) An HR-DSSS encoding method that encodes four or eight bits into one symbol.

Composite signal A signal composed of more than one sine wave.
Concurrent client A client running the same time as another client of the same process.

Concurrent server A server that can process many requests at the same time and share its time between many requests.

Congestion avoidance In Frame Relay, a method using two bits that explicitly notify the source and destination of congestion.
Congestion control A method to manage network and internetwork traffic to improve throughput.

Congestion Excessive network or internetwork traffic causing a general degradation of service.

Connecting device A tool that connects computers or networks.

Connection control The technique used by the transport layer to deliver segments.
Connection establishment The preliminary setup necessary for a logical connection prior to actual data transfer.

Connection termination A message sent to end a connection.

Connectionless iterative server A connectionless server that processes one request at a time.

Connectionless service A service for data transfer without connection establishment or termination.

Connection-oriented concurrent server A connection-oriented server that can serve many clients at the same time.

Connection-oriented service A service for data transfer involving establishment and termination of a connection.

Constant bit rate (CBR) The data rate of an ATM service class that is designed for customers requiring real-time audio or video services.

Constellation diagram A graphical representation of the phase and amplitude of different bit combinations in digital-to-analog modulation.

Consultative Committee for International Telegraphy and Telephony (CCITT) An international standards group now known as the ITU-T. **Contention** An access method in which two or more devices try to transmit at the same time on the same channel.

Control connection The FTP connection used for control information (commands and responses).

Controlled access A multiple access method in which the stations consult one another to determine who has the right to send.

Convergence sublayer (CS) In ATM protocol, the upper AAL sublayer that adds a header or a trailer to the user data.

Cookie A string of characters that holds some information about the client and must be returned to the server untouched.

Core The glass or plastic center of an optical fiber.

Core-Based Tree (CBT) In multicasting, a group-shared protocol that uses a center router as the root of the tree.

Country domain A subdomain in the Domain Name System that uses two characters as the last suffix.

CRC checker The process that validates the CRC remainder.

CRC generator The process that creates the CRC remainder.

Critical angle In refraction, the value of the angle of incidence that produces a 90-degree angle of refraction.

Crossbar switch A switch consisting of a lattice of horizontal and vertical paths. At the intersection of each horizontal and vertical path, there is a crosspoint that can connect the input to the output.

Crosspoint The junction of an input and an output on a crossbar switch. **Crosstalk** The noise on a line caused by signals traveling along another line.
Cryptography The science and art of transforming messages to make them secure and immune to attacks.

Cyclic code A linear cod in which the cyclic shifting (rotation) of each codeword creates another code word.

CSNET A network sponsored by the National Science Foundation

riginally intended for universities.

Cycle The repetitive unit of a periodic signal.

Cyclic redundancy check (CRC) A highly accurate error-detection method based on interpreting a pattern of bits as a polynomial.

Data element The smallest entity that can represent a piece of information. A bit.

Data connection The FTP connection used for data transfer.

Data encryption standard (DES) The U.S. government standard encryption method for nonmilitary and nonclassified use.

Data exchange protocol A protocol that uses the secret key to encrypt the data for secrecy and to encrypt the message digest for integrity.
Data level The number of different symbols used to represent a digital signal.
Data link connection identifier (DLCI) A number that identifies the virtual circuit in Frame Relay.

Data link control The responsibilities of the data link layer: flow control and error control.

Data link layer The second layer in the Internet model. It is responsible for node-to-node delivery.

Data Over Cable System Interface Specifications (DOCSIS) A standard for data transmission over an HFC network.

Data rate The number of data elements sent in one second.

Data transfer phase The intermediate phase in circuit-switched or virtual-circuit network in which data transfer takes place.

Data transparency See *transparency*.

Datagram approach (to packet switching) A data transmission method in which each data unit is independent of others.

Datagram In packet switching, an independent data unit.

Datagram network A packet-switched network in which packets are independent from each other.

Dataword The smallest block of data in block coding.

Datagram socket A structure designed to be used with a connectionless protocol such as UDP.

DC component See *direct current*.

De facto standard A protocol that has not been approved by an organized body but adopted as a standard through widespread use.

De jure standard A protocol that has been legislated by an officially recognized body.

Deadlock A situation in which a task cannot proceed because it is waiting for an even that will never occur.

Decibel (dB) A measure of the relative strength of two signal points.
Decryption Recovery of the original message from the encrypted data.

Default mask The mask for a network that is not subnetted.
Default routing A routing method in which a router is assigned to receive all packets with no match in the routing table.

efense Advanced Research Projects Agency (DARPA) A government organization, which, under the name of ARPA funded ARPANET and the Internet.
Delta modulation An analog-to-digital conversion technique in which the value of the digital signal is based on the difference between the current and the previous sample values.

Delayed response strategy A technique used by IGMP to prevent unnecessary traffic on a LAN.

Demodulation The process of separating the carrier signal from the information-bearing signal.

Demodulator A device that performs demodulation.

Demultiplexer (DEMUX) A device that separates a multiplexed signal into its original components.

Denial of service attack A form of attack in which the site is flooded with so many phony requests that is eventually forced to deny service.
Dense wave-division multiplexing (DWDM) A WDM method that can multiplex a very large number of channels by spacing channels closer together.
Destination-unreachable message An ICMP error-reporting message sent to a source when a router cannot route a datagram or a host cannot deliver a datagram.

Dibit A unit of data consisting of two bits.

Differential Manchester encoding A digital-to-digital polar encoding method that features a transition at the middle of the bit interval as well as an inversion at the beginning of each 1 bit.

Differentiated Services (DS or Diffserv) A class-based QoS model designed for IP.
Diffie-Hellman protocol A key management protocol that provides a one-time session key for 2 parties.

Digest A condensed version of a document.

digital AMPS (D-AMPS) A second-generation cellular phone system that is a digital version of AMPS.

Digital data Data represented by discrete values or conditions.

Digital data service (DDS) A digital version of an analog leased line with a rate of 64 Kbps.

Digital service unit (DSU) A device that allows the connection of a user's device to a digital line.

Digital signal (DS) service A telephone company service featuring a hierarchy of digital signals.

Digital signal A discrete signal with a limited number of values.
Digital signature A method to authenticate the sender of a message.

Digital subscriber line (DSL) A technology using existing telecommunication networks to accomplish high-speed delivery of data, voice, video, and multimedia.

Digital subscriber line access multiplexer (DSLAM) A telephone company site device that functions like an ADSL modem.
Digital-to-analog conversion The representation of digital information by an analog signal.

Digital-to-digital conversion The representation of digital information by a digital signal.

Ddigital-to-analog modulation The representation of digital information by an analog signal.

Digital-to-digital encoding The representation of digital information by a digital signal.

Dijkstra's algorithm In link state routing, an algorithm that finds the shortest path to other routers.

Direct current (DC) A zero-frequency signal with a constant amplitude.
Direct delivery A delivery in which the final destination of the packet is a host connected to the same physical network as the sender.
Direct sequence spread spectrum (DSSS) A wireless transmission method in which each bit to be sent by the sender is replaced by a sequence of bits called a chip code.

Discard eligibility (DE) A bit that defines that a packet can be discarded if there is congestion in the network.

Discrete cosine transform (DCT) A JPEG phase in which a transformation changes the 64 values so that the relative relationships between pixels are kept but the redundancies are revealed.
Discrete multitone technique (DMT) A modulation method combining elements of QAM and FDM.

Distance Vector Multicast Routing Protocol (DVMRP) A protocol based on distance vector routing that handles multicast routing in conjunction with IGMP.
Distance vector routing A routing method in which each router sends its neighbors a list of networks it can reach and the distance to that network.
Distortion Any change in a signal due to noise, attenuation, or other influences.
Distributed coordination fucntion (DCF) The basic access method in wireless LANs; stations contend with each other to get access to the channel.
Distributed database Information stored in many locations.
Distributed interframe space (DIFS) In wireless LANs, a period of time that a station waits before sending a control frame.

Distributed processing A strategy in which services provided for the network reside at multiple sites.

Distribution hub In an HFC network, a site that modulates and distributes signals.
DNS server A computer that holds information about the name space.
Domain A subtree of the domain name space.

Domain name In the DNS, a sequence of labels separated by dots.
Domain name space A structure for organizing the name space in which the names are defined in an inverted-tree structure with the root at the top.
Domain Name System (DNS) A TCPIP application service that converts user-friendly names to IP addresses.

Dotted-decimal notation A notation devised to make the IP address easier to read; each byte is converted to its decimal equivalent and then set off from its neighbor by a decimal.

Downlink Transmission from a satellite to an earth station.

Downloading Retrieving a file or data from a remote site.

Downstream data band In an HFC network, the 550 to 750 MHz band for data from the Internet to the subscriber premises.

Dual stack Two protocols (IPv4 and IPv6) on a station.

Duplex mode See *full-duplex mode.*

Dynamic document A Web document created by running a CGI program at the server site.

Dynamic Domain Name System (DDNS) A method to update the DNS master file dynamically.

Dynamic Host Configuration Protocol (DHCP) An extension to BOOTP that dynamically assigns configuration information.

Dynamic mapping A technique in which a protocol is used for address resolution.
Dynamic routing Routing in which the routing table entries are updated automatically by the routing protocol.

E lines The European equivalent of T lines.

Echo-request and reply message An ICMP query message that determines whether two systems (hosts or routers) can communicate with each other.

Effective bandwidth The bandwidth that the network needs to allocate for the flow of traffic; a function of three values: average data rate, peak data rate, and maximum burst size.

Electromagnetic spectrum The frequency range occupied by lectromagnetic energy.

Electronic code block (ECB) mode ADES and triple DES operation method in which a long message is divided into 64-bit blocks before being encrypted separately.

Electronic mail (email) A method of sending messages electronically based on mailbox addresses rather than a direct host-to-host exchange.

Electronics Industries Association (EIA) An organization that promotes electronics manufacturing concerns. It has developed interface standards such as EIA-232, EIA-449, and EIA-530.

Email See *electronic mail.*

Encapsulating Security Payload (ESP) A protocol defined by IPSec that provides privacy as well as a combination of integrity and message authentication.
Encapsulation The technique in which a data unit from one protocol is placed within the data field portion of the data unit of another protocol.
Encryption Converting a message into an unintelligible form that is unreadable unless decrypted.

End office A switching office that is the terminus for the local loops.
End system A sender or receiver of data.

Ephemeral port number A port number used by the client.

Error control The detection and handling of errors in data transmission.
Error correction by retransmission The process of correcting bits by resending the data.

Error-reporting message An ICMP message sent to the source to report an error.

ESS-transition mobility A station in a wireless LAN that can move from one ESS to another.

Establishing state In PPP, a state in which communication begins and options are negotiated.

Ethernet A local area network using CSMACD access method. See *IEEE Project 802.3.*

Even parity An error-detection method in which an extra bit is added to the data unit so that the total number of 1s becomes even.

Excess burst size In Frame Relay, the maximum number of bits in excess of *Bc* that the user can send during a predefined period of time.

Extended Service Set (ESS) A wireless LAN service composed of two or more BSSs with APs as defined by the IEEE 802.11 standard.
Extension header Extra headers in the IPv6 datagram that provide additional functionality.

Exterior routing Routing between autonomous systems.
External link LSA A message that announces all the networks outside the AS.
Extranet A private network that uses the TCPIP protocol suite that allows authorized access from outside users.

Fast Ethernet Ethernet with a data rate of 100 Mbps.

Fast retransmission Retransmission of a segment in TCP protocol when three acknowledgements have been received that imply the loss or corruption of that segment.

Fast Ethernet See *100Base-T.*

Federal Communications Commission (FCC) A government agency that regulates radio, television, and telecommunications.

Fiber distributed data interface (FDDI) A high-speed (100-Mbps) LAN, defined by ANSI, using fiber optics, dual ring topology, and the token-passing access method. Today an FDDI network is also used as a MAN.
Fiber link Ethernet Ethernet using fiber optic media.

Fiber node In an HFC network, the location of the optical fiber and coaxial fiber juncture.

Fiber-optic cable A high-bandwidth transmission medium that carries data signals in the form of pulses of light. It consists of a thin cylinder of glass or plastic, called the core, surrounded by a concentric layer of glass or plastic called the cladding.

File Transfer Protocol (FTP) In TCPIP, an application layer protocol that transfers files between two sites.

Filtering A process in which a bridge makes forwarding decisions.
Finite state machine A machine that goes through a limited number of states.
Firewall A device (usually a router) installed between the internal network of an organization and the rest of the Internet to provide security.
First-in, first-out (FIFO) queue A queue in which the first item in is the first item out.

Flag A bit pattern or a character added to the beginning and the end of a fame to separate the frames.

Flag field In an HDLC frame, an 8-bit synchronization sequence that identifies the beginning or end of a frame.

Flat name space A method to map a name to an address in which there is no hierarchical structure.

Flooding Saturation of a network with a message.

Flow control A technique to control the rate of flow of frames (packets or messages).
Flow label An IPv6 mechanism to enable the source to request special handling of a packet.

Footprint An area on Earth that is covered by a satellite at a specific time.
Forward error correction Correction of errors at the receiver.
Forward explicit congestion notification (FECN) A bit in the Frame Relay packet that notifies the destination of congestion.
Forwarding Placing the packet in its route to its destination.
Forwarding port A port on a bridge that forwards a received frame.
Fourier analysis The mathematical technique used to obtain the frequency spectrum of an aperiodic signal if the time-domain representation is given.
Fragmentation offset A field in the IP header used in fragmentation to show the relative position of the fragment with respect to the whole

datagram.
Fragmentation The division of a packet into smaller units to accommodate a protocol's MTU.

Frame A group of bits representing a block of data.

Frame bursting A technique in CSMACD Gigabit Ethernet in which multiple frames are logically connected to each other to resemble a longer frame.
Frame check sequence (FCS) The HDLC error-detection field containing either a 2- or 4-byte CRC.

Frame Relay A packet-switching specification defined for the first two layers of the Internet model. There is no network layer. Error checking is done on end-to-end basis instead of on each link.

Frame Relay assemblerdisassembler (FRAD) A device used in Frame Relay to handle frames coming from other protocols.

Frequency The number of cycles per second of a periodic signal.

Framing bit A bit used for synchronization purposes in TDM.

Frequency division multiple access (FDMA) A multiple access method in which the bandwidth is divided into channels.

Frequency hopping spread spectrum (FHSS) A wireless transmission method in which the sender transmits at one carrier frequency for a short period of time, then hops to another carrier frequency for the same amount of time, hops again for the same amount of time, and so on. After *N* hops, the cycle is repeated.

Frequency modulation (FM) An analog-to-analog modulation method in which the carrier signal's frequency varies with the amplitude of the modulating signal.

Frequency shift keying (FSK) A digital-to-analog encoding method in which the frequency of the carrier signal is varied to represent binary 0 or 1.

Frequency-division multiple access (FDMA) An access method technique in which multiple sources use assigned bandwidth in a data communication band.

Frequency The number of cycles per second of a periodic signal.
Frequency-division multiplexing (FDM) The combining of analog signals into a single signal.

Frequency-domain plot A graphical representation of a signal's frequency components.
Full-duplex mode A transmission mode in which communication can be two way simultaneously.

Full-duplex switched Ethernet Ethernet in which each station, in its own separate collision domain, can both send and receive.

Fully qualified domain name (FQDN) A domain name consisting of labels beginning with the host and going back through each level to the root node.

Fundamental frequency The frequency of the dominant sine wave of a composite signal.

Gatekeeper In the H.323 standard, a server on the LAN that plays the role of the registrar server.

Gateway A device used to connect two separate networks that use different communication protocols.

General header A part of an HTTP request or response message that gives general information about the message.

Generic domain A subdomain in the domain name system that uses generic suffixes.

Geographical routing A routing technique in which the entire address space is divided into blocks based on physical landmasses.

Geosynchronous Earth orbit An orbit that allows a satellite to remain fixed above a certain spot on earth.

Gigabit Ethernet Ethernet with a 1000 Mbps data rate.

Gigabit medium independent interface (GMII) In Gigabit Ethernet, a specification that defines how the reconciliation sublayer is to be connected to the transceiver.

Global Internet The Internet.

Global Positioning System (GPS) An MEO public satellite system consisting of 24 satellites and used for land and sea navigation. GPS is not used for communications.

Global System for Mobile Communication (GSM) A second-generation cellular phone system used in Europe.

Globalstar An LEO satellite system with 48 satellites in six polar orbits with each orbit hosting eight satellites.

Go-Back-N ARQ An error-control method in which the frame in error and all following frames must be retransmitted.

Grafting Resumption of multicast messages.

Ground propagation Propagation of radio waves through the lowest portion of the atmosphere (hugging the earth).

Group An analog signal created by 12 voice channels multiplexed together.
Group membership Belonging to a group.

Group-shared tree A multicast routing feature in which each group in the system shares the same tree.

Guard band A bandwidth separating two signals.

Guided media Transmission media with a physical boundary.

H.323 A standard designed by ITU to allow telephones on the public telephone network to talk to computers (called terminals in H.323) onnected to the Internet.

Half-duplex mode A transmission mode in which communication can be two-way but not at the same time.

Hamming code A method that adds redundant bits to a data unit to detect and correct bit errors.

Hamming distance The number of differences between the corresponding bits in two datawords.

Handoff Changing to a new channel as a mobile device moves from one cell to another.

Handshake protocol A protocol to establish or terminate a connection. **Harmonics** Components of a digital signal, each having a different amplitude, frequency, and phase.

Hash function An algorithm that creates a fixed-size digest from a variable-length message.

Hashed-message authentication code (HMAC) A MAC based on a keyless hash function such as SHA-1.

Head end A cable TV office.

Header Control information added to the beginning of a data packet. Also, in an email, the part of the message that defines the sender, the receiver, the subject of the message, and other information.

Header translation Conversion of the IPv6 header to IPv4.

Hertz (Hz) Unit of measurement for frequency.

Hexadecimal colon notation In IPv6, an address notation consisting of 32 hexadecimal digits, with every four digits separated by a colon.

Hierarchical name space A name space made of several parts, with each succeeding part becoming more and more specific.

Hierarchical routing A routing technique in which the entire address space is divided into levels based on specific criteria.

High bit rate digital subscriber line (HDSL) A service similar to the T1-line that can operate at lengths up to 3.6 km.

High Rate Direct Sequence Spread Spectrum (HR-DSSS) A signal generation method similar to DSSS except for the encoding method (CCK). **High-level Data Link Control (HDLC)** A bit-oriented data link protocol defined by the ISO. It is used in X.25 protocol. A subset, called link access procedure (LAP), is used in other protocols. It is also a base for many data link protocols used in LANs.

Homepage A unit of hypertext or hypermedia available on the Web that is the main page for an organization or an individual.

Hop count The number of nodes along a route. It is a measurement of distance in routing algorithms.

Hop limit An IPv6 field that limits the number of routers that a packet can visit.
Hop-to-hop delivery Transmission of frames from one node to the next.
Horn antenna A scoop-shaped antenna used in terrestrial microwave communication.
Host A station or node on a network.

Host file A file, used when the Internet was small, that mapped host names to host addresses.

Hostid The part of an IP address that identifies a host.

Host-specific routing A routing method in which the full IP address of a host is given in the routing table.

Hub A central device in a star topology that provides a common connection among the nodes.

Huffman encoding A statistical compression method using variable-length codes to encode a set of symbols.

Hybrid network A network with a private internet and access to the global Internet.
Hybrid-fiber-coaxial (HFC) network The second generation of cable networks; uses fiber optic and coaxial cable.

Hypermedia Information containing text, pictures, graphics, and sound that are linked to other documents through pointers.

Hypertext Information containing text that is linked to other documents through pointers.

HyperText Markup Language (HTML) The computer language for specifying the contents and format of a web document. It allows additional text to include codes that define fonts, layouts, embedded graphics, and hypertext links.

HyperText Transfer Protocol (HTTP) An application service for retrieving a web document.

Idle state In PPP, a state in which the link is inactive.

Image file In FTP, the default format for transferring binary files. The file is sent as continuous streams of bits without any interpretation or encoding.
Incumbent local exchange carrier (ILEC) A telephone company that provided services before 1996 and is the owner of the cabling system.
Indirect delivery A delivery in which the source and destination of a packet are in different networks.

Infrared wave A wave with a frequency between 300 GHz and 400 THz; usually used for short-range communications.

Inner product A number produced by multiplying two sequences, element by element, and summing the products.

Institute of Electrical and Electronics Engineers (IEEE) A group consisting of professional engineers which has specialized societies whose committees prepare standards in members' areas of specialty.

Integrated Services (IntServ) A flow-based QoS model designed for IP.

Integrity A data quality of being noncorrupted.

Interactive audiovideo Real-time communication with sound and images. **Interautonomous system routing protocol** A protocol to handle ransmissions between autonomous systems.

Interdomain routing Routing among autonomous systems. **Interexchange carrier (IXC)** A long-distance company that, prior to the Act of 1996, provided communication services between two customers in different LATAs.

Interface The boundary between two pieces of equipment. It also refers to mechanical, electrical, and functional characteristics of the connection. **Interference** Any undesired energy that interferes with the desired signals. **Interframe space (IFS)** In wireless LANs, a time interval between two frames to control access to the channel.

Interim Standard 95 (IS-95) One of the dominant second-generation cellular telephony standards in North America.

Interior routing Routing inside an autonomous system.

Interleaving Taking a specific amount of data from each device in a regular order.

International Organization of Standardization (ISO) A worldwide organization that defines and develops standards on a variety of topics. **International Telecommunications Union-Telecommunication tandardization Sector (ITU-T)** A standards organization formerly known as the CCITT.

internet A collection of networks connected by internetworking devices such as routers or gateways.

Internet A global internet that uses the TCPIP protocol suite. **Internet address** A 32-bit or 128-bit network-layer address used to uniquely define a host on an internet using the TCPIP protocol.

Internet Architecture Board (IAB) The technical adviser to the ISOC; oversees the continuing development of the TCPIP protocol suite. **Internet Assigned Numbers Authority (IANA)** A group supported by the U.S. government that was responsible for the management of Internet domain names and addresses until October 1998.

Internet Control Message Protocol (ICMP) A protocol in the TCPIP protocol suite that handles error and control messages.

Internet Control Message Protocol, version 6 (ICMPv6) A protocol in IPv6 that handles error and control messages.

Internet Corporation for Assigned Names and Numbers (ICANN) A private, nonprofit corporation managed by an international board that assumed IANA operations.

Internet draft A working Internet document (a work in progress) with no official status and a six-month lifetime.

Internet Engineering Steering Group (IESG) An organization that oversees the activity of IETF.

Internet Engineering Task Force (IETF) A group working on the design and development of the TCPIP protocol suite and the Internet.

Internet Group Management Protocol (IGMP) A protocol in the TCPIP protocol suite that handles multicasting.

Internet Key Exchange (IKE) A protocol designed to create security associations in SADBs.

Internet Mail Access Protocol, version 4 (IMAP4) A complex and powerful protocol to handle the transmission of electronic mail.
Internet Mobile Communication for year 2000 (ITM-2000) An ITU issued blueprint that defines criteria for third generation cellular telephony.
Internet model A 5-layer protocol stack that dominates data communications and networking today.

Internet Network Information Center (INTERNIC) An agency responsible for collecting and distributing information about TCPIP protocols.
Internet Protocol (IP) The network-layer protocol in the TCPIP protocol suite governing connectionless transmission across packet switching networks.
Internet Protocol next generation (IPng) See Internet Protocol version 6 (IPv6).
Internet Protocol version 4 (IPv4) The current version of Internet Protocol.
Internet Protocol, version 6 (IPv6) The sixth version of the Internetworking Protocol; it features major IP addressing changes.

Internet Research Task Force (IRTF) A forum of working groups focusing on long-term research topics related to the Internet.

Internet Security Association and Key Management Protocol (ISAKMP) A protocol designed by the national Security Agency (NSA) that actually implements the exchanges defined in IKE.

Internet service provider (ISP) Usually, a company that provides Internet services.
Internet Society (ISOC) The nonprofit organization established to publicize the Internet.

Internet standard A thoroughly tested specification that is useful to and adhered to by those who work with the Internet. It is a formalized regulation that must be followed.

Internetwork Protocol Control Protocol (IPCP) In PPP, the set of protocols that establish and terminate a network layer connection for IP packets.
Internetwork (internet) A network of networks.

Internetworking Connecting several networks together using internetworking devices such as routers and gateways.

Intracoded frame (I-frame) An independent frame that is not related to any other frame and appearing at regular intervals.

Intranet A private network that uses the TCPIP protocol suite.
Inverse domain A subdomain in the DNS that finds the domain name given the IP address.

Inverse multiplexing Taking data from one source and breaking it into portions that can be sent across lower-speed lines.

IP datagram The Internetworking Protocol data unit.

IP Security (IPSec) A collection of protocols designed by the IETF (Internet Engineering Task Force) to provide security for a packet carried on the Internet.

IrDA port A port that allows a wireless keyboard to communicate with a PC.
Iridium A 66-satellite network that provides communication from any Earth site to another.

ISDN user port (ISUP) A protocol at the upper layer of SS7 that provides services similar to those of an ISDN network.

Isochronous transmission A type of transmission in which the entire stream of bits is synchronized under the control of a common clock.
Iterative resolution Resolution of the IP address in which the client may send its request to multiple servers before getting an answer.

Iterative server In the client-server model, a server that can serve only one client at a time.

jamming signal In CSMACD, a signal sent by the first station that detects collision to alert every other station of the situation.

Java A programming language used to create active Web documents.
Jitter A phenomenon in real-time traffic caused by gaps between consecutive packets at the receiver.

Joint Photographic Experts Group (JPEG) A standard for compressing continuous-tone picture.

Jumbo group An analog signal created by six multiplexed master groups.

Karn's Algorithm An algorithm that does not include the retransmitted segments in calculation of round-trip time.

Keepalive message A message that establishes a relationship between the two routers.

Keepalive timer A timer that prevents a long idle connection between two TCPs.
Kerberos An authentication protocol used by Windows 2000.
Key A number that a cipher operates on.

Key distribution center (KDC) In secret key encryption, a trusted third party that shares a key with each user.

LAN emulation (LANE) Local area network emulation using ATM switches.
LAN emulation client (LEC) In ATM LANs, client software that receives services from a LES.

LAN emulation server (LES) In ATM LANs, server software that creates a virtual circuit between the source and destination.

Layered architecture A model based on ordered tiers.

Leaky bucket algorithm An algorithm to shape bursty traffic.

Least-cost tree An MOSPF feature in which the tree is based on a chosen metric instead of shortest path.

Leave report An IGMP message sent by a host when no process is interested in a specific group.

Legacy ATM LAN LAN in which ATM technology is used as a backbone to connect traditional LANs.

Line coding Converting binary data into signals.

Linear block code A block code in which adding two codewords creates another codeword.

Line-of-sight propagation The transmission of very high frequency signals in straight lines directly from antenna to antenna.

Link Control Protocol (LCP) A PPP protocol responsible for establishing, maintaining, configuring, and terminating links.

Link local address An OPv6 address used by a private LAN.

Link state advertisement (LSA) In OSPF, a method that disperses information.
link state database In link state routing, a database common to all routers and made from LSP information.

Link state packet (LSP) In link state routing, a small packet containing routing information sent by a router to all other routers.

Link state routing A routing method in which each router shares its knowledge of changes in its neighborhood with all other routers.

Link state update packet A packet that provides information about a specific route or routes.

Link The physical communication pathway that transfers data from one device to another.

Load The number of packets sent to a network.

Local access and transport area (LATA) An area covered by one or more telephone companies.

Local access Using a terminal directly connected to the computer. **Local address** The part of an email address that defines the name of a special file, called the user mailbox, where all of the mail received for a user is stored for retrieval by the user agent.

Local area network (LAN) A network connecting devices inside a single building or inside buildings close to each other.

Local area network emulation (LANE) Software that enables an ATM switch to behave like a LAN switch.

Local call service A telephone service handling local calls, usually charging a flat monthly fee.

Local exchange carrier (LEC) A telephone company that handles services inside a LATA.

Local Internet service provider The same as an Internet service provider.
Local ISP The same as an Internet service provider.

Local loop The link that connects a subscriber to the telephone central office.
Local management information (LMI) A protocol used in Frame Relay to provide. management features.

Logical address An address defined in the network layer.

Logical link control (LLC) The upper sublayer of the data link layer as defined by IEEE Project 802.2.

Logical Link Control and Adaptation Protocol (L2CAP) A Bluetooth layer used for data exchange on an ACL link.

Logical tunnel The encapsulation of a multicast packet inside a unicast packet to enable multicast routing by non-multicast routers.
Longest mask matching The technique in CIDR in which the longest prefix is handled first when searching a routing table.

low Earth orbit (LEO) A polar satellite orbit with an altitude between 500 and 2000 km. A satellite with this orbit has a rotation period of 90 to 120 minutes.
Low-pass channel A channel that passes frequencies between 0 and *f*.
Mail transfer agent (MTA) An SMTP component that transfers the mail across the Internet.
Management Information Base (MIB) The database used by SNMP that holds the information necessary for management of a network.
Manchester encoding A digital-to-digital polar encoding method in which a transition occurs at the middle of each bit interval for the purpose of synchronization.
Man-in-the-middle attack A key management problem in which an intruder intercepts and sends messages between the intended sender and receiver.
Mapped address An IPv6 address used when a computer that has migrated to Ipv6 wants to send a packet to a computer still using IPv4.
Mask For IPv4, a 32-bit binary number that gives the first address in the block (the network address) when ANDed with an address in the block.
Master The one Bluetooth station in a piconet that controls all the others.
Master group An analog signal created by 10 multiplexed supergroups.
Maximum burst size The maximum length of time traffic is generated at the peak rate.

Maximum transfer unit (MTU) The largest size data unit a specific network can handle.

Media player A help application that plays an audiovideo file; used by a browser.
Media server A server accessed by a media player to download an audiovideo file.

Medium access control (MAC) sublayer The lower sublayer in the data link layer defined by the IEEE 802 project. It defines the access method and access control in different local area network protocols.

Medium attachment unit (MAU) See *transceiver*

Medium bandwidth The difference between the highest and lowest frequencies a medium can support.

Medium dependent interface (MDI) In Fast Ethernet, implementation-specific hardware that connects the transceiver to the medium.

Medium Earth orbit (MEO) A satellite orbit positioned between the two Van Allen belts. A satellite at this orbit takes six hours to circle the earth.

Medium independent interface (MII) In Fast Ethernet hardware that connects an external transceiver to the reconciliation layer.

Membership report An IGMP message sent by a host or router interested in joining a specific group.

Mesh topology A network configuration in which each device has a dedicated point-to-point link to every other device.

Message access agent (MAA) A client-server program that pulls the stored email messages.

Message authentication A security measure in which the sender of the message is verified for every message sent.

Message authentication code A keyed hash function.

Message transfer agent (MTA) An SMTP component that transfers the message across the Internet.

Metric A cost assigned for passing through a network.

Metropolitan area network (MAN) A network that can span a geographical area the size of a city.

Minimum Hamming distance In a set of words, the smallest Hamming distance between all possible pairs.

Microwave Electromagnetic waves ranging from 2 GHz to 40 GHz.
Minislot In an HFC network, a time slot for timesharing of the upstream channels.
Mixer A device that combines real-time signals from different sources into one signal.

Mobile host A host that can move from one network to another.

Mobile switching center (MSC) In cellular telephony, a switching office that coordinates communication between all base stations and the telephone central office.

Mobile telephone switching office (MTSO) An office that controls and coordinates communication between all of the cell offices and the telephone control office.

Modem A device consisting of a modulator and a demodulator. It converts a digital signal into an analog signal (modulation) and vice versa (demodulation).
Modification detection code (MDC) The digest created by a hash function.
Modular arithmetic Arithmetic that uses a limited range of integers (O to n-1).
Modulation Modification of one or more characteristics of a carrier wave by an information-bearing signal.

Modulus The upper limit in modular arithmetic (n).

Modulator A device that converts a digital signal to an analog signal suitable for transmission across a telephone line.

Monoalphabetic substitution An encryption method in which each occurrence of a character is replaced by another character in the set.
Motion picture experts group (MPEG) A method to compress videos.

MT-RJ A fiber-optic cable connector.

multicast address An address used for multicasting.

multicast backbone (MBONE) A set of internet routers supporting multicasting through the use of tunneling.

Multicast Open Shortest Path First (MOSPF) A multicast protocol that uses multicast link state routing to create a source-based least cost tree.

Multicast router A router with a list of loyal members related to each router interface that distributes the multicast packets.

Multicasting A transmission method that allows copies of a single packet to be sent to a selected group of receivers.

Multihoming service A service provided by SCTP that allows a computer to be connected to different networks.

Multiline transmission, 3-level (MLT-3) encoding A line coding scheme featuring 3 levels of signals and transitions at the beginning of the 1 bit.

Multimode graded-index fiber An optical fiber with a core having a graded index of refraction.

Multimode step-index fiber An optical fiber with a core having a uniform index of refraction. The index of refraction changes suddenly at the corecladding boundary.

Multiple access (MA) A line access method in which every station can access the line freely.

Multiple unicasting Sending multiple copies of a message, each with a different unicast address.

Multiplexer (MUX) A device used for multiplexing.

Multiplexing The process of combining signals from multiple sources for transmission across a single data link.

Multiplicative decrease A congestion avoidance technique in which the threshold is set to half of the last congestion window size, and the congestion window size starts from one again.

Multipurpose Internet Mail Extension (MIME) A supplement to SMTP that allows non-ASCII data to be sent through SMTP.

Multistage switch An array of switches designed to reduce the number of crosspoints.
Multistream service A service provided by SCTP that allows data transfer to be carried using different streams.

N2 problem A problem due to the large number of keys needed in symmetric key distribution.

Nagle's algorithm An algorithm that attempts to prevent silly window syndrome at the sender's site; both the rate of data production and the network speed are taken into account.

Name space All the names assigned to machines on an internet.

Name-address resolution Mapping a name to an address or an address to a name.

National service provider (NSP) A backbone network created and maintained by a specialized company.

Needham-Schroeder protocol A key management protocol using multiple challenge-response interactions between 2 entities.

Negative acknowledgment (NAK) A message sent to indicate the rejection of received data.

Netid The part of an IP address that identifies the network.

Network A system consisting of connected nodes made to share data, hardware, and software.

Nnetwork access point (NAP) A complex switching station that connects backbone networks.

Network address An address that identifies a network to the rest of the Internet; it is the first address in a block.

Network address translation (NAT) A technology that allows a private network to use a set of private addresses for internal communication and a set of global Internet addresses for external communication.

Network allocation vector (NAV) In CSMACA, the amount of time that must pass before a station can check the line for idleness.

Network Control Protocol (NCP) In PPP, a set of control protocols that allows the encapsulation of data coming from network layer protocols.

Network interface card (NIC) An electronic device, internal or external to a station, that contains circuitry to enable the station to be connected to the network.
Network layer The third layer in the Internet model, responsible for the delivery of a packet to the final destination.

Network link LSA An LSA packet that announces the existence of all of the routers connected to the network.

Network Virtual Terminal (NVT) A TCPIP application protocol that allows remote login.

Networking state A PPP state in which packets of user data and packets for control are transmitted.

Network-specific routing Routing in which all hosts on a network share one entry in the routing table.

Network-to-network interface (NNI) In ATM, the interface between two networks.
Next header In IPv6, an 8-bit field defining the header that follows the base header in the datagram.

ext-hop routing A routing method in which only the address of the next hop is listed in the routing table instead of a complete list of the stops the packet must make.

Node An addressable communication device (e.g., a computer or router) on a network.

Node-to-node delivery Transfer of a data unit from one node to the next.
Noise Random electrical signals that can be picked by the transmission medium and result in degradation or distortion of the data.

Noiseless channel An error-free channel.

Noisy channel A channel that can produce error in data transmission.
Nonce A large random number that is used once to distinguish a fresh authentication request from a used one.

Nonpersistent connection A connection in which one TCP connection is made for each requestresponse.

Nonpersistent strategy A random multiple access method in which a station waits a random period of time after a collision is sensed.
Nonrepudiation A security aspect in which a receiver must be able to prove that a received message came from a specific sender.

Nonreturn to zero (NRZ) A digital-to-digital polar encoding method in which the signal level is always either positive or negative.

Nonreturn to zero, invert (NRZ-I) An NRZ encoding method in which the signal level is inverted each time a 1 is encountered.

Nonreturn to zero, level (NRZ-L) An NRZ encoding method in which the signal level is directly related to the bit value.

Normal response mode (NRM) In HDLC, a communication mode in which the secondary station must have permission from the primary station before transmission can proceed.

Nyquist bit rate The data rate based on the Nyquist theorem.

Notification message A BGP message sent by a router whenever an error condition is detected or a router wants to close the connection.

No-transition mobility In wireless LANs, mobility confined inside a BSS or non-mobility.
Nyquist theorem A theorem that states that the number of samples needed to adequately represent an analog signal is equal to twice the highest frequency of the original signal.

Oakley A key creating protocol, developed by Hilarie Orman, which is one of the three components of IKE protocol.

Odd parity An error-detection method in which an extra bit is added to the data unit such that the sum of all 1-bits becomes odd.
Omnidirectional antenna An antenna that sends out or receives signals in all directions.

On-demand audiovideo Another name for streaming stored audiovideo.

One's complement A representation of binary numbers in which the complement of a number is found by complementing all bits.

Open message A BGP message sent by a router to create a neighborhood relationship.

Open shortest path first (OSPF) An interior routing protocol based on link state routing.

Open system A model that allows two different systems to communicate regardless of their underlying architecture.

Open Systems Interconnection (OSI) model A seven-layer model for data communication defined by ISO.

Open-loop congestion control Policies applied to prevent congestion.
Optical carrier (OC) The hierarchy of fiber-optic carriers defined in SONET. The hierarchy defines up to 10 different carriers (OC-1, OC-3, OC-12, . . . , OC-192), each with a different data rate.

Optical fiber A thin thread of glass or other transparent material to carry light beams.

Optional attribute A BGP path attribute that need not be recognized by every router.

Orbit The path a satellite travels around the earth.

Orthogonal Frequency Division Multiplexing (OFDM) A multiplexing method similar to FDM, with all the subbands used by one source at a given time.

Orthogonal sequence A sequence with special properties between elements.
Otway-Rees protocol A key management protocol with less steps than the Needham-Schroeder method.

Out-of-band signaling Using two separate channels for data and control.
Output feedback (OFB) mode A mode similar to the CFB mode with one difference. Each bit in the ciphertext s independent of the previous bit or bits.
Overhead Extra bits added to the data unit for control purposes.

Packet switching Data transmission using a packet-switched network.

Packet Synonym for data unit, mostly used in the network layer.

Packet-filter firewall A firewall that forwards or blocks packets based on the information in the network-layer and transport-layer headers.

Packet-switched network A network in which data are transmitted in independent units called packets.

Page A unit of hypertext or hypermedia available on the Web.
Parabolic dish antenna An antenna shaped like a parabola used for terrestrial microwave communication.

Parallel transmission Transmission in which bits in a group are sent simultaneously, each using a separate link.

Parameter-problem message An ICMP message that notifies a host that there is an ambiguous or missing value in any field of the datagram.
Parity bit A redundant bit added to a data unit (usually a character) for error checking.

Parity check An error-detection method using a parity bit.

Partially qualified domain name (PQDN) A domain name that does not include all the levels between the host and the root node.
Passive open The state of a server as it waits for incoming requests from a client.

Password Authentication Protocol (PAP) A simple two-step authentication protocol used in PPP.

Path layer A SONET layer responsible for the movement of a signal from its optical source to its optical destination.

Path overhead Control information used by the SONET path layer.
Path The channel through which a signal travels.

Path vector routing A routing method on which BGP is based; in this method, the ASs through which a packet must pass are explicitly listed.

P-box A hardware circuit used in encryption that connects input to output.

Peak amplitude The maximum signal value of a sine wave.

Peak data rate The maximum data rate of the traffic.

Ppeer-to-peer process A process on a sending and a receiving machine that communicate at a given layer.

Per hop behavior (PHB) In the Diffserv model, a 6-bit field that defines the packet-handling mechanism for the packet.

Period The amount of time required to complete one full cycle.

Periodic signal A signal that exhibits a repeating pattern.

Permanent virtual circuit (PVC) A virtual circuit transmission method in which the same virtual circuit is used between source and destination on a continual basis.

Persistence timer A technique to handle the zero window-size advertisement.
Persistent connection A connection in which the server leaves the connection open for more requests after sending a response.

Persistent strategy In CSMA, a strategy in which the station sends a frame after sensing the line.

Personal Communication System (PCS) A generic term for a commercial cellular system that offers several kinds of communication services.
Phase modulation (PM) An analog-to-analog modulation method in which the carrier signal's phase varies with the amplitude of the modulating signal.
Phase shift keying (PSK) A digital-to-analog modulation method in which the phase of the carrier signal is varied to represent a specific bit pattern.
Phase The relative position of a signal in time.

PHY sublayer The transceiver in Fast Ethernet.

Physical address The address of a device used at the data link layer (MAC address).

Physical layer signaling (PLS) sublayer An Ethernet sublayer that encodes and decodes data.

Physical layer The first layer of the Internet model, responsible for the mechanical and electrical specifications of the medium.

Physical topology The manner in which devices are connected in a network.
Piconet A Bluetooth network.

Piggybacking The inclusion of acknowledgment on a data frame.

Pipelining In Go-Back-*n* ARQ, sending several frames before news is received concerning previous frames.

Pixel A picture element of an image.

Plain old telephone system (POTS) The conventional telephone network used for voice communication.

Plaintext In encryptiondecryption, the original message.

Playback buffer A buffer that stores the data until they are ready to be played.
Point coordination function (PCF) In wireless LANs, an optional and complex access method implemented in an infrastructure network.
Point of presence (POP) A switching office where carriers can interact with each other.

Point-to-point access See *point-to-point connection*.

Point-to-point connection A dedicated transmission link between two devices.
Point-to-point link A dedicated transmission link between two devices.
Point-to-Point Protocol (PPP) A protocol for data transfer across a serial line.
Poison reverse A feature added to split horizon in which a table entry that has come through on interface is set to infinity in the update packet.
Polar encoding A digital-to-analog encoding method that uses two levels (positive and negative) of amplitude.

Policy routing A path vector routing feature in which the routing tables are based on rules set by the network administrator rather than a metric.
Poll In the primarysecondary access method, a procedure in which the primary station asks a secondary station if it has any data to transmit.
Pollfinal (PF) bit A bit in the control field of HDLC; if the primary is sending, it can be a poll bit; if the secondary is sending, it can be a final bit.
Pollselect An access method protocol using poll and select procedures. See *poll*. See *select*.

Polling An access method in which one device is designated as a primary station and the others as the secondary stations. The access is controlled by the primary station.

Polyalphabetic substitution An encryption method in which each occurrence of a character can have a different substitute.

Polynomial An algebraic term that can represent a CRC divisor.

Port In a URL, the port number of the server.

Port address In TCPIP protocol an integer identifying a process.
Port number An integer that defines a process running on a host.
Post Office Protocol, version 3 (POP3) A popular but simple SMTP mail access protocol.

P-persistent A CSMA persistence strategy in which a station sends with probablity p if it finds the line idle.

P-persistent strategy A CSMA persistence strategy in which a station sends with probability *p* if it finds the line idle.

Preamble The 7-byte field of an IEEE 802.3 frame consisting of alternating 1s and 0s that alert and synchronize the receiver.

Predicted frame (P-frame) An MPEG frame which contains only the changes from the preceding frame.

Prefix The common part of an address range.

Predictive encoding In audio compression, encoding only the differences between the samples.

presentation layer The sixth layer of the OSI model responsible for translation, encryption, authentication, and data compression.

Pretty Good Privacy (PGP) A protocol that provides all four aspects of security in the sending of email.

Primary server A server that stores a file about the zone for which it is an authority.
Primary station In primarysecondary access method, a station that issues commands to the secondary stations.

Priority queueing A queuing technique in which packets are assigned to a priority class, each with its own queue.

Privacy A security aspect in which the message makes sense only to the intended receiver.

Private key In conventional encryption, a key shared by only one pair of devices, a sender and a receiver. In public-key encryption, the private key is known only to the receiver.

Private network A network that is isolated from the Internet.

Process A running application program.

Process-to-process delivery Delivery of a packet from the sending process to the destination process.

Product block A combination of P-boxes and S-boxes to get a more complex cipher block.

Project 802 The project undertaken by the IEEE in an attempt to solve LAN incompatibility. See also *IEEE Project 802*.

Propagation speed The rate at which a signal or bit travels; measured by distancesecond.
Propagation time The time required for a signal to travel from one point to another.
Protocol Independent Multicast (PIM) A multicasting protocol family with two members, PIM-DM and PIM-SM; both protocols are unicast-protocol dependent.
Protocol Independent Multicast, Dense Mode (PIM-DM) A source-based routing protocol that uses RPF and pruninggrafting strategies to handle multicasting.
Protocol Independent Multicast, Sparse Mode (PIM-SM) A group-shared routing protocol that is similar to CBT and uses a rendezvous point as the source of the tree.

Protocol Rules for communication.

Protocol suite A stack or family of protocols defined for a complex communication system.

Proxy ARP A technique that creates a subnetting effect; on server answers ARP requests for multiple hosts.

Proxy firewall A firewall that filters a message based on the information available in the message itself (at the application layer).

Proxy server A computer that keeps copies of responses to recent requests.
Pruning Stopping the sending of multicast messages from an interface.
Pseudoheader Information from the IP header used only for checksum calculation in UDP and TCP packet.

Pseudorandom noise (PN) A pseudorandom code generator used in FHSS.
Public key In public-key encryption, a key known to everyone.
Public key infrastructure (PKI) A hierarchical structure of CA servers.
Public-key cryptography A method of encryption based on a nonreversible encryption algorithm. The method uses two types of keys:

he public key is known to the public; the private key (secret key) is known only to the receiver.

Pulse amplitude modulation (PAM) A technique in which an analog signal is sampled; the result is a series of pulses based on the sampled data.
Pulse code modulation (PCM) A technique that modifies PAM pulses to create a digital signal.

Pulse stuffing In TDM, a technique that adds dummy bits to the input lines with lower rates.

Pure ALOHA The original ALOHA.

Pulse rate The number of symbols per second.

Quadbit A unit of data consisting of four bits.

Quadrature amplitude modulation (QAM) A digital-to-analog modulation method in which the phase and amplitude of the carrier signal vary with the modulating signal.

Quality of service (QoS) A set of attributes related to the performance of the connection.

Quantization The assignment of a specific range of values to signal amplitudes.
Query message An ICMP message that helps a host or a network manager get specific information from a router or another host. Or, an IGMP message that requests group information from a router or a host. Or, a DNS message that requests information.

Queue A waiting list.

Radio wave Electromagnetic energy in the 3-KHz to 300-GHz range.
Random access A medium access category in which each station can access the medium without being controlled by any other station.
Ranging In an HFC network, a process that determines the distance between the CM and the CMTS.

Rate adaptive asymmetrical digital subscriber line (RADSL) A DSL-based technology that features different data rates depending on the type of communication.
Raw socket A structure designed for protocols that directly use the services of IP and use neither stream sockets nor datagram sockets.
Read-only memory (ROM) Permanent memory with contents that cannot be changed.

Real-Time Streaming Protocol (RTSP) An out-of-band control protocol designed to add more functionality to the streaming audiovideo process.
Real-time Transport Control Protocol (RTCP) A companion protocol to RTP with messages that control the flow and quality of data and allow the recipient to send feedback to the source or sources.

Real-time Transport Protocol (RTP) A protocol for real-time traffic; used in conjunction with UDP.

Receiver window In the TCP sliding window protocol, the window at the receiver site.

Reconciliation sublayer A Fast Ethernet sublayer which passes data in 4-bit format to the MII.

Recursive resolution Resolution of the IP address in which the client sends its request to a server that eventually returns a response.

Redirection An ICMP message type that informs the sender of a preferred route.
Redundancy The addition of bits to a message for error control.

Reed-Solomon A complex, but efficient, cyclic code.

Reflection The phenomenon related to the bouncing back of light at the boundary of two media.

Refraction The phenomenon related to the bending of light when it passes from one medium to another.

Regenerator A device that regenerates the original signal from a corrupted signal. See also *repeater*.

Regional cable head (RCH) In an HFC network, the main distribution site.
Regional ISP A small ISP that is connected to one or more NSPs.
Registrar An authority to register new domain names.

Registrar server In SIP, a server that knows the IP address of the callee.
Reliability A QoS flow characteristic; dependability of the transmission.
Remote access Using a terminal that is not directly connected to a computer.
Remote bridge A device that connects LANs and point-to-point networks; often used in a backbone network.

Remote host The computer that a user wishes to access while seated physically at another computer.

Remote server A program run at a site physically removed from the user.
Rendezvous router A router that is the core or center for each multicast group; it becomes the root of the tree.

Rendezvous-point tree A group-shared tree method in which there is one tree for each group.

Repeater A device that extends the distance a signal can travel by regenerating the signal.

Replay attack The resending of a message that has been intercepted by an intruder.

Request for Comment (RFC) A formal Internet document concerning an Internet issue.

Request header A part of the HTTP request message that specifies the client's configuration and the client's preferred document format.
Resolver The DNS client that is used by a host that needs to map an address to a name or a name to an address.

Resource Reservation Protocol (RSVP) A signaling protocol to help IP create a flow and make a resource reservation to improve QoS.
Retransmission time-out The expiration of a timer that controls the retransmission of packets.

Response header A part of the HTTP response message that specifies the server's configuration and special information about the request.
Response message A DNS message type that returns information.

Retransmission timer A timer that controls the waiting time for an acknowledgment of a segment.

Return to zero (RZ) A digital-to-digital encoding technique in which the voltage of the signal is zero for the second half of the bit interval.
Reuse factor In cellular telephony, the number of cells with a different set of frequencies.

Reverse Address Resolution Protocol (RARP) A TCPIP protocol that allows a host to find its Internet address given its physical address.
Reverse path broadcasting (RPB) A technique in which the router forwards only the packets that have traveled the shortest path from the source to the router.

Reverse path forwarding (RPF) A technique in which the router forwards only the packets that have traveled the shortest path from the source to the router.
Reverse path multicasting (RPM) A technique that adds pruning and grafting to RPB to create a multicast shortest path tree that supports dynamic membership changes.

Rijndael algorithm An algorithm named after its two Belgian inventors, Vincent Rijmen and Joan Daemen that is the basis of AES.

Ring topology A topology in which the devices are connected in a ring. Each device on the ring receives the data unit from the previous device, regenerates it, and forwards it to the next device.

Rivest, Shamir, Adleman (RSA) encryption See *RSA encryption*.
RJ45 A coaxial cable connector.

Roaming In cellular telephony, the ability of a user to communicate outside of his own service provider's area.

Root server In DNS, a server whose zone consists of the whole tree. A root server usually does not store any information about domains but delegates its authority to other servers, keeping references to those servers.
Rotary dialing Accessing the switching station through a phone that sends a digital signal to the end office.

Rotation cipher A keyed or keyless cipher in which the input bits are rotated to the left or right to create output bits.
Round-trip time (RTT) The time required for a datagram to go from a source to a destination and then back again.

Route A path traveled by a packet.

Router An internetworking device operating at the first three OSI layers. A router is attached to two or more networks and forwards packets from one network to another.

Router link LSA An LSA packet that advertises all of the links of a router.
Router-solicitation and advertisement message An ICMP message sent to obtain and disperse router information.

Routing Information Protocol (RIP) A routing protocol based on the distance vector routing algorithm.

Routing table A table containing information a router needs to route packets. The information may include the network address, the cost, the address of the next hop, and so on.

Routing The process performed by a router; finding the next hop for a datagram.
RSA encryption A popular public-key encryption method developed by Rivest, Shamir, and Adleman.

Sampling rate The number of samples obtained per second in the sampling process.

Sampling The process of obtaining amplitudes of a signal at regular intervals.
Satellite network A combination of nodes that provides communication form one point on the earth to another.

S-box An encryption device made of decoders, P-boxes, and encoders.
Scatternet A combination of piconets.

Scrambling In digital-to-digital conversion, modifying part of the rules in line coding scheme to create bit synchronization.

Search algorithm A rule for finding the next hop.

Secondary server In DNS, a server that transfers the complete information about a zone from another server (primary or secondary) and stores the file on its local disk.

Secondary station In pollselect access method, a station that sends a response in answer to a command from a primary station.

Secret-key encryption A security method in which the key for encryption is the same as the key for decryption; both sender and receiver have the same key.

Secure Hash Algorithm 1 (SHA-1) A hash algorithm designed by the National Institute of Standards and Technology (NIST). It was published as a Federal Information Processing Standard (FIPS).

Secure Socket Layer (SSL) A protocol designed to provide security and compression services to data generated from the application layer.

Security Association (SA) An IPSec signaling protocol that creates a logical connection between 2 hosts.

Security association database (SADB) A database defining a set of single security associations.

Security parameter index (SPI) A parameter that uniquely distinguish on security association from the others.

Security The protection of a network from unauthorized access, viruses, and catastrophe

Segment The packet at the TCP layer. Also, the length of transmission medium shared by devices.

Segmentation and reassembly (SAR) The lower AAL sublayer in the ATM protocol in which a header andor trailer may be added to produce a 48-byte element.

Segmentation The splitting of a message into multiple packets; usually performed at the transport layer.

Select In pollselect access method, a procedure in which the primary station asks a secondary station if it is ready to receive data. **Selective-repeat ARQ** An error-control method in which only the frame in error is resent.

Self-synchronization Synchronization of long strings of 1s or 0s through the coding method.

Semantics The meaning of each section of bits.

Sender window In the TCP sliding window protocol, the window at the sender site.

Sequence number The number that denotes the location of a frame or packet in a message.

Serial transmission Transmission of data one bit at a time using only one single link.

Server A program that can provide services to other programs, called clients.
Server control point (SCP) In SS7 terminology, the node that controls the whole operation of the network.

Service-point address See *port address.*

Session Initiation Protocol (SIP) In voice over IP, an application protocol that establishes, manages, and terminates a multimedia session.

Session layer The fifth layer of the OSI model, responsible for the establishment, management, and termination of logical connections between two end users.

Setup phase In virtual circuit switching, a phase in which the source and destination use their global addresses to help switches make table entries for the connection.

S-frame An HDLC frame used for supervisory functions such as acknowledgment, flow control, and error control; it contains no user data.
Shannon capacity The theoretical highest data rate for a channel.

Shielded twisted-pair (STP) Twisted-pair cable enclosed in a foil or mesh shield that protects against electromagnetic interference.

Shift cipher The simplest monoalphabetic cipher in which the plaintext and ciphertext consist of letters. In the encryption algorithm, the characters are shifted down the character list; in the decryption algorithm, the characters are shifted up the character list.

Shift register A register in which each memory location, at a time click, accepts the bit at its input port, stores the new bit, and displays it on the output port.

Short interframe space (SIFS) In CSMACA, a period of time that the destination waits after receiving the RTS.

Shortest path tree A routing table formed by using the Dijkstra algorithm.
Signal Electromagnetic waves propagated along a transmission medium.
Signaling connection control point (SCCP) In SS7, the control points used for special services such as 800 calls.

Signal element The shortest section of a signal (time-wise) that represents a data element.

Signal point (SP) In SS7 terminology, the user telephone or computer is connected to the signal points.

Signal rate The number of signal elements sent in one second.

Signal transport port (STP) In SS7 terminology, the node used by the signaling network.

Signaling System Seven (SS7) The protocol that is used in the signaling network.
Signal level The number of values allowed in a particular signal.

Signal-to-noise ratio (SNR) The signal strength divided by the noise, both in decibels.

Silly window syndrome A situation in which a small window size is advertised by the receiver and a small segment sent by the sender.

Simple and efficient adaptation layer (SEAL) An AAL layer designed for the Internet (AAL5).

Simple bridge A networking device that links two segments; requires manual maintenance and updating

Simple Mail Transfer Protocol (SMTP) The TCPIP protocol defining electronic mail service on the Internet.

Simple Network Management Protocol (SNMP) The TCPIP protocol that specifies the process of management in the Internet.

Simple Protocol The simple protocol we used to show an access method without flow and error control.

Simplex mode A transmission mode in which communication is one way.
Sine wave An amplitude-versus-time representation of a rotating vector.
Single-bit error Error in a data unit in which only one single bit has been altered.
Single-mode fiber An optical fiber with an extremely small diameter that limits beams to a few angles, resulting in an almost horizontal beam.

Site local address An IPv6 address for a site having several networks, but not connected to the Internet.

SKEME A protocol for key exchange designed by Hugo Krawcyzk. It is one of the three protocols that form the basis of IKE.

Sky propagation Propagation of radio waves into the ionosphere and then back to earth.

Slash notation A shorthand method to indicate the number of 1s in the mask.
Slave In a piconet, a station under control of a master.

Sliding window A protocol that allows several data units to be in transition before receiving an acknowledgment.

Sliding window ARQ An error-control protocol using sliding window concept.
Slotted ALOHA The modified ALOHA access method in which time is divided into slots and each station is forced to start sending data only at the beginning of the slot.

Slow convergence A RIP shortcoming apparent when a change somewhere in the internet propagates very slowly through the rest of the internet.
Slow start A congestion-control method in which the congestion window size increases exponentially at first.

Socket address A structure holding an IP address and a port number.
Socket An end point for a process; two sockets are needed for

communication.

Socket interface An API based on UNIX that defines a set of system calls (procedures) that are an extension of system calls used in UNIX to access files.

Solicited response A RIP response sent only in answer to a request.

Source address (SA) The address of the sender of the message.

Source quench A method, used in ICMP for flow control, in which the source is advised to slow down or stop the sending of datagrams because of congestion.

Source routing bridge A source or destination station that performs some of the duties of a transparent bridge as a method to prevent loops.

Source routing Explicitly defining the route of a packet by the sender of the packet.

Source-based tree A tree used for multicasting by multicasting protocols in which a single tree is made for each combination of source and group.

Source quench message An ICMP message sent to slow down or stop the sending of datagrams.

Source-to-destination delivery The transmission of a message from the original sender to the intended recipient.

Space propagation A type of propagation that can penetrate the ionosphere.

Space-division switching Switching in which the paths are separated from each other spatially.

Spanning tree A tree with the source as the root and group members as leaves; a tree that connects all of the nodes.

Spanning tree algorithm An algorithm that prevents looping when two LANs are connected by more than one bridge.

Spatial compression Compressing an image by removing redundancies.

Special-query message An IGMP query message sent by a router to ensure that no host or router is interested in continuing membership in a group.

Specific host on this network A special address in which the netid is all 0s and the hostid is explicit.

Spectrum The range of frequencies of a signal.

Split horizon A method to improve RIP stability in which the router selectively chooses the interface from which updating information is sent.

Spread spectrum A wireless transmission technique that requires a bandwidth several times the original bandwidth.

Standard Ethernet The conventional Ethernet operating at 10 mbps.

Star backbone A backbone in which the logical topology is a star.

Star topology A topology in which all stations are attached to a central device (hub).

Start bit In asynchronous transmission, a bit to indicate the beginning of transmission.

Start frame delimiter (SFD) A 1-byte field in the IEEE 802.3 frame that signals the beginning of the readable (nonpreamble) bit stream.

State transition diagram A diagram to illustrate the states of a finite state machine.

Static document On the World Wide Web, a fixed-content document that is created and stored in a server.

Static mapping A technique in which a list of logical and physical address correspondences is used for address resolution.

Static routing A type of routing in which the routing table remains unchanged.

Stationary host A host that remains attached to one network.

Statistical TDM A TDM technique in which slots are dynamically allocated to improve efficiency.

Status line In the HTTP response message a line that consists of the HTTP version, a space, a status code, a space, a status phrase.

Stop bit In asynchronous transmission, one or more bits to indicate the end of transmission.

Stop-and-wait ARQ An error-control protocol using stop-and-wait flow control.

Stop-and-Wait Protocol A protocol in which the sender sends on frame, stops until it receives confirmation from the receiver, and then sends the next frame.

Store-and-forward switch A switch that stores the frame in an input buffer until the whole packet has arrived.

Straight tip connector A type of fiber-optic cable connector using a bayonet locking system.

Stream Control Transmission Protocol (SCTP) The transport layer protocol designed for Internet telephony and related applications.

Stream socket A structure designed to be used with a connection-oriented protocol such as TCP.

Streaming live audiovideo Broadcast data from the Internet that a user can listen to or watch.

Streaming stored audiovideo Data downloaded as files from the Internet that a user can listen to or watch.

Strong collision Creating two message with the same digest.

Structure of Management Information (SMI) In SNMP, a component used in network management.

STS multiplexer/demultiplexer A SONET device that multiplexes and demultiplexes signals.

Stub link A network that is connected to only one router.

Subnet subnetwork.
Subnet address The network address of a subnet

Subnet mask The mask for a subnet.

Subnetwork A part of a network.

Subscriber channel connector A fiber-optic cable connector using a pushpull locking mechanism.

Substitution cipher A bit-level encryption method in which *n* bits substitute for another *n* bits as defined by P-boxes, encoders, and decoders.

Suffix For a network, the varying part (similar to the hostid) of the address. In DNS, a string used by an organization to define its host or resources.

Summary link to AS boundary router LSA An LSA packet that lets a router inside an area know the route to an autonomous boundary router. **Summary link to network LSA** An LSA packet that finds the cost of reaching networks outside of the area.

Supergroup A signal composed of five multiplexed groups.

Supernet A network formed from two or more smaller networks. **Supernet mask** The mask for a supernet.

Supervisory frame See *S-frame*.

Switch A device connecting multiple communication lines together.

Switched Ethernet An Ethernet in which a switch, replacing the hub, can direct a transmission to its destination.

Switched virtual circuit (SVC) A virtual circuit transmission method in which a virtual circuit is created and in existence only for the duration of the exchange.
Switched56 A temporary 56-Kbps digital connection between two users.

Switching office The place where telephone switches are located.

Symmetric digital subscriber line (SDSL) A DSL-based technology similar to HDSL, but using only one single twisted-pair cable.

Symmetric key The key used for both encryption and decryption. **Symmetric-key cryptography** A cipher in which the same key is used for encryption and decryption.

Synchronization points Reference points introduced into the data by the session layer for the purpose of flow and error control.

Synchronous connection oriented (SCO) link In a Bluetooth network, a physical link created between a master and a slave that reserves specific slots at regular intervals.

Synchronous Digital Hierarchy (SDH) The ITU-T equivalent of SONET.

Synchronous Optical Network (SONET) A standard developed by ANSI for fiber optic technology that can transmit high-speed data. It can be used to deliver text, audio, and video.

Synchronous payload envelope (SPE) The part of the SONET frame containing user data and transmission overhead.

Synchronous TDM A TDM technique in which each input has an allotment in the output even when it is not sending data.

Synchronous transmission A transmission method that requires a constant timing relationship between the sender and the receiver.

Synchronous transport module (STM) A signal in the SDH hierarchy.
Synchronous transport signal (STS) A signal in the SONET hierarchy.
Syndrome A sequence of bit generated by applying the error checking function to a codeword.

Syntax The structure or format of data, meaning the order in which they are presented.

T lines A hierarchy of digital lines designed to carry speech and other signals in digital forms. The hierarchy defined T-1, T-2, T-3, and T-4 lines.
Tandem office The toll office in a telephone network.

Tag A formatting instruction embedded in an HTML document.

TCP timer The timers used by TCP to handle retransmission, zero window-size advertisements, long idle connections, and connection termination.
TCPIP protocol suite A group of hierarchical protocols used in an internet.
TDM bus A time-division switch in which the input and output lines are connected to a high-speed bus through microswitches.

Teardown phase In virtual circuit switching, the phase in which the source and destination inform the switch to erase their entry.

Telecommunications Exchange of information over distance using electronic equipment.

Teleconferencing Audio and visual communication between remote users.
Teledesic A system of satellites that provides fiber-optic communication (broadband channels, low error rate, and low delay)

Telephone user port (UTP) A protocol at the upper layer of SS7 that is responsible for setting up voice calls.

TELNET See *Terminal Network*.

Temporal compression An MPEG compression method in which redundant frames are removed.

Ten-Gigabit Ethernet The new implementation of Ethernet operating at 10 Gbps.
Terminal Network (TELNET) A general purpose client-server program that allows remote login.

Three-way handshaking A sequence of events for connection establishment or termination consisting of the request, then the acknowledgment of the request, and then confirmation of the acknowledgment.

Terminating state A PPP state in which several packets are exchanged between the two ends for house cleaning and closing the link.

Terminator An electronic device that prevents signal reflections at the end of a cable.

Thick Ethernet See *10Base5.*

Thin Ethernet See *10Base2.*

Three-layer switch A switch at the network layer; a router.

Three-way handshake A sequence of events for connection establishment or termination consisting of the request, then the acknowledgment of the request, and then confirmation of the acknowledgment.

Throughput The number of bits that can pass through a point in one second.
Ticket An encrypted message containing a session key.

Ticket-granting server (TGS) A Kerberos server that issues tickets.

Time division duplexing TDMA (TDD-TDMA) In a Bluetooth network, a kind of half-duplex communication in which the slave and receiver send and receive data, but not at the same time (half-duplex).

Time division multiple access (TDMA) A multiple access method in which the bandwidth is just one time-shared channel.

Time to live (TTL) The lifetime of a packet.

Time-division multiplexing (TDM) The technique of combining signals coming from low-speed channels to share time on a high-speed path.

Time-division switching A circuit-switching technique in which time-division multiplexing is used to achieve switching.

Time-domain plot A graphical representation of a signal's amplitude versus time.

Time-exceeded message An ICMP message sent to inform a source that (1) its datagram has a time-to-live value of zero, or (2) the fragments of a message have not been received within a set time limit.

Time-slot interchange (TSI) A time-division switch consisting of RAM and a control unit.

Timestamp An IP header option used to record the time of datagram processing by a router. Also, a method to handle jitter in interactive real-time audiovideo.

Timestamp-request and reply message An ICMP message sent to determine the round-trip time or to synchronize clocks.

Time-waited timer A TCP timer used in connection termination that allows late segments to arrive.

T-lines A hierarchy of digital lines designed to carry speech and other signals in digital forms. The hierarchy defines T-1, T-2, T-3, and T-4 lines.

Token A small packet used in token-passing access method.
Token bucket An algorithm that allows idle hosts to accumulate credit for the future in the form of tokens.

Token passing An access method in which a token is circulated in the network. The station that captures the token can send data.

Token Ring A LAN using a ring topology and token-passing access method.

Toll call service An inter-LATA or intra-LATA telephone service charged to the caller.

Topology The structure of a network including physical arrangement of devices.
Touch-tone dialing A telephone dialing method in which each key is represented by two small bursts of analog signals.

Traffic Messages on a network.

Traffic control A method for shaping and controlling traffic in a wide area network.
Traffic shaping A mechanism to control the amount and the rate of the traffic sent to the network to improve QoS.

Trailer Control information appended to a data unit.

Transaction capablities application port (TCAP) A protocol at the upper layer of SS7 that provides remote procedure calls that let an application program on a computer invoke a procedure on another computer.

Transceiver A device that both transmits and receives.

Transceiver cable In Ethernet, the cable that connects the station to the transceiver. Also called the attachment unit interface.

Transient link A network with several routers attached to it.

Transition state The different phases through which a PPP connection goes.
Translation Changing from one code or protocol to another.

Transmission Control Protocol (TCP) A transport protocol in the TCPIP protocol suite.

Transmission Control ProtocolInternetworking Protocol (TCPIP) A five-layer protocol suite that defines the exchange of transmissions across the Internet.

Transmission medium The physical path linking two communication devices.
Transmission path (TP) In ATM, the physical connection between two switches.
Transmission rate The number of bits sent per second.

Transparency The ability to send any bit pattern as data without it being mistaken for control bits.

Transparent bridge Another name for a learning bridge.

Transparent data Data that can contain control bit patterns without being interpreted as control.

Transport Layer Security (TLS) A security protocol at the transport level designed to provide security on the WWW.

Transport layer The fourth layer in the Internet and OSI model;

responsible for reliable end-to-end delivery and error recovery.

Transpositional cipher A character-level encryption method in which the position of the character changes.

Trellis-coded modulation A modulation technique that includes error correction.
Trilateration A two-dimensional method of finding a location given the distances from 3 different points.

Triangulation A two-dimensional method of finding a location given the distances from 3 different points.

Tribit A unit of data consisting of three bits.

Triple DES An algorithm compatible with DES that uses three DES blocks and two 56-bit keys.

Trivial File Transfer Protocol (TFTP) An unreliable TCPIP protocol for file transfer that does not require complex interaction between client and server.
Trunk Transmission media that handle communications between offices.

Tunneling In multicasting, a process in which the multicast packet is encapsulated in a unicast packet and then sent through the network. In VPN, the encapsulation of an encrypted IP datagram in a second outer datagram. For IPv6, a strategy used when two computers using IPv6 want to communicate with each other when the packet must pass through a region that uses IPv4.

Twisted-pair cable A transmission medium consisting of two insulated conductors in a twisted configuration.

Twisted-pair Ethernet An Ethernet using twisted-pair cable; 10Base-T.

Two-dimensional parity check An error detection method in two dimensions.

Two-layer switch A bridge with many ports and a design that allows better (faster) performance.

Type of service (TOS) A criteria or value that specifies the handling of the datagram.
U-frame An HDLC unnumbered frame carrying link management information.
Unbalanced configuration An HDLC configuration in which one device is primary and the others secondary.

Unguided medium A transmission medium with no physical boundaries.

Unicast address An address belonging to one destination.

Unicast message A message sent to just one destination.

Unicast routing The sending of a packet to just one destination.

Unicasting The sending of a packet to just one destination.

Unicode The international character set used to define valid characters in computer science.

Unidirectional antenna An antenna that sends or receives signals in one direction.
Uniform Resource Locator (URL) A string of characters (address) that identifies a page on the World Wide Web.

Unipolar encoding A digital-to-digital encoding method in which one nonzero value represents either 1 or 0; the other bit is represented by a zero value.

Unnumbered frame See *U-frame.*

Unshielded twisted-pair (UTP) A cable with wires that are twisted together to reduce noise and crosstalk. See also *twisted-pair cable* and *shielded twisted-pair.*

Unspecified bit rate (UBR) The data rate of an ATM service class specifying only best-effort delivery.

Update message A BGP message used by a router to withdraw destinations that have been advertised previously or to announce a route to a new destination.

Uplink Transmission from an earth station to a satellite.

Uploading Sending a local file or data to a remote site.

Upstream data band In an HFC network, the 5 to 42 MHz band for data from the subscriber premises to the Internet.

User agent (UA) An SMTP component that prepares the message, creates the envelope, and puts the message in the envelope.

User authentication A security measure in which the sender identity is verified before the start of a communication.

User Datagram Protocol (UDP) A connectionless TCPIP transport layer protocol.
User datagram The name of the packet in the UDP protocol.

User network interface (UNI) The interface between a user and the ATM network.
User support layers The session, presentation, and application layers.
User-to-network interface (UNI) In ATM, the interface between an end point (user) and an ATM switch.

V series ITU-T standards that define data transmission over telephone lines. Some common standards are V.32, V32bis, V.90, and V92.

Variable bit rate (VBR) The data rate of an ATM service class for users needing a varying bit rate.

Very high bit rate digital subscriber line (VDSL) A DSL-based technology for short distances.

Video Recording or transmitting of a picture or a movie.
Video band In an HFC network, the band from 54 to 550 MHz for downstream video.

Vigenere cipher A polyalphabetic substitution scheme that uses the position of a character in the plaintext and the character's position in the alphabet.
Virtual channel identifier (VCI) A field in an ATM cell header that defines a channel.

Virtual circuit (VC) A logical circuit made between the sending and receiving computer. The connection is made after both computers do handshaking. After the connection, all packets follow the same route and arrive in sequence.

Virtual circuit approach to packet switching A packet switching method in which all packets of a message or session follow the exact same route.

Virtual circuit identifier (VCI) A field in an ATM cell header that defines a channel.
Virtual circuit switching A switching technique used in switched WANs.

Virtual connection identifier A VCI or VPI.

Virtual link An OSPF connection between two routers that is created when the physical link is broken. The link between them uses a longer path that probably goes through several routers.

Virtual local area network (VLAN) A technology that divides a physical LAN into virtual workgroups through software methods.

Virtual path (VP) In ATM, a connection or set of connections between two switches.
Virtual path identifier (VPI) A field in an ATM cell header that identifies a path.
Virtual path identifiervirtual channel identifier (VPIVCI) Two fields used together to route an ATM cell.

Virtual private network (VPN) A technology that creates a network that is physically public, but virtually private.

Virtual tributary (VT) A partial payload that can be inserted into a SONET frame and combined with other partial payloads to fill out the frame.

Voice Over Frame Relay (VOFR) A Frame Relay option that can handle voice data.

Voice over IP A technology in which the Internet is used as a telephone network.
Walsh table In CDMA, a two-dimensional table used to generate orthogonal sequences.

Wavelength The distance a simple signal can travel in one period.

Wave-division multiplexing (WDM) The combining of modulated light signals into one signal.

Weak collision Given a digest, creating a second message with the same digest.

Web page A unit of hypertext or hypermedia available on the Web.

Web Synonym for World Wide Web (WWW).

Weighted fair queueing A packet scheduling technique to improve QoS in which the packets are assigned to queues based on a given priority number.

Well-known port number A port number that identifies a process on the server.

Well-known attribute Path information that every BGP router must recognize.

Well-known port A port number that identifies a process on the server.

Wide area network (WAN) A network that uses a technology that can span a large geographical distance.

Wide area telephone service (WATS) A telephone service in which the charges are based on the number of calls made.

Window size field The size of the sliding window used in flow control.

Wireless communication Data transmission using unguided media.

Wireless LAN A LAN which uses unguided media.

World Wide Web (WWW) A multimedia Internet service that allows users to traverse the Internet by moving from one document to another via links that connect them together.

X.25 An ITU-T standard that defines the interface between a data terminal device and a packet-switching network.

X.509 An ITU-T standard for public key infrastructure (PKI).

Zone In DNS, what a server is responsible for or has authority over.