GOVERNMENT ARTS AND SCIENCE COLLEGE
(Affiliated to Manonmaniam Sundaranar University,
Tirunelveli)
PALKULAM, KANYAKUMARI- 629 401

STUDY MATERIAL FOR BCA

COMPUTER NETWORKING

SEMESTER - VI

ACADEMIC YEAR 2022 – 2023

PREPARED BY

DEPARTMENT OF COMPUTER SCIENCE

UNIT – I

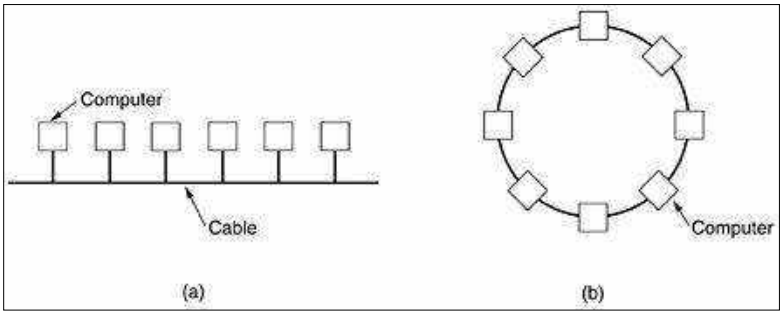
NETWORK HARDWARE & SOFTWARE

Local Area Networks

Local area networks, generally called LANs, are privately-owned networks within a single building or campus of up to a few kilometers in size. They are widely used to **connect** personal computers and workstations in company offices and factories to share resources (e.g., printers) and exchange information. LANs are distinguished from other kinds of networks by three characteristics: (1) their size, (2) their transmission technology, and (3) their topology.

LANs are restricted in size, which means that the worst-case transmission time is **bounded** and known in advance. Knowing this bound makes it possible to use certain kinds of designs that would not otherwise be possible. It also simplifies network management.

Two broadcast networks. (a) Bus. (b) Ring.



LANs may use a transmission technology consisting of a cable to which all the machines are attached, like the telephone company party lines once used in rural areas. Traditional **LANs** run at speeds of 10 Mbps to 100 Mbps, have low delay (microseconds or nanoseconds), and make very few errors. Newer LANs operate at up to 10 Gbps. In this book, we will adhere to tradition and measure line speeds in megabits/sec (1 Mbps is 1,000,000 bits/sec) and gigabits/sec (1 Gbps is 1,000,000,000 bits/sec).

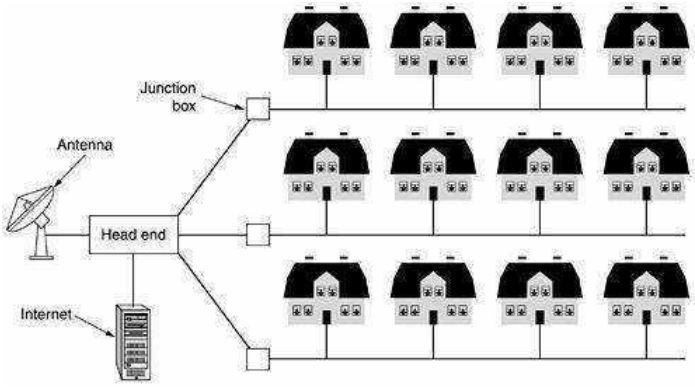
A second type of broadcast system is the ring. In a ring, each bit propagates around **on** its own, not waiting for the rest of the packet to which it belongs. Typically, each bit circumnavigates the entire ring in the time it takes to transmit

a few bits, often before the complete packet has even been transmitted. As with all other broadcast systems, some rule is needed for arbitrating simultaneous accesses to the ring. Various methods, such as having the machines take turns, are in use. IEEE 802.5 (the IBM token ring), is a ring-based LAN operating at 4 and 16 Mbps. FDDI is another example of a ring network.

Broadcast networks can be further divided into static and dynamic, depending on how the channel is allocated. A typical static allocation would be to divide time **into** discrete intervals and use a round-robin algorithm, allowing each machine to broadcast only when its time slot comes up. Static allocation wastes channel capacity when a machine has nothing to say during its allocated slot, so most systems attempt to allocate the channel dynamically (i.e., on demand).

Dynamic allocation methods for a common channel are either centralized or decentralized. In the centralized channel allocation method, there is a single entity, for example, a bus arbitration unit, which determines who goes next. It might do this by accepting requests and making a decision according to some internal algorithm. In the decentralized channel allocation method, there is no central entity; each machine must decide for itself whether to transmit. You might think that this always leads to chaos, but it does not. Later we will study many algorithms designed to bring order out of the potential chaos.

A metropolitan area network based on cable TV.

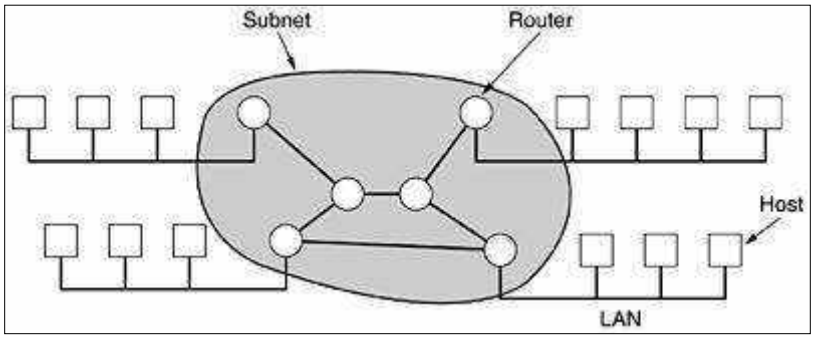


A metropolitan area network, or MAN, covers a city. The best-known example of a MAN is the **cable** television network available in many cities. This system grew from earlier community antenna systems used in areas with poor

over-the-air television reception. In these early systems, a large antenna was placed on top of a nearby hill and signal was then piped to the subscribers' houses.

At first, **these** were locally-designed, ad hoc systems. Then companies began jumping into the business, getting contracts from city governments to wire up an entire city. The next step was television programming and even entire channels designed for cable only. Often these channels were highly specialized, such as all news, all sports, all cooking, all gardening, and so on. But from their inception until the late 1990s, they were intended for television reception only.

Wide Area Networks



A wide area network, or WAN, spans a large geographical area, often a country or continent. It contains of machines intended for running user (i.e., application) programs. We will follow traditional usage and call these machines hosts. The hosts are connected by a communication subnet, or just subnet for short. The hosts are owned by the customers (e.g., people's personal computers), whereas the communication subnet is typically owned and operated by a telephone company or Internet service provider. The job of the subnet is to carry messages from host to host, just as the telephone system carries words from speaker to listener. Separation of the pure communication aspects of the network (the subnet) from the application aspects (the hosts), greatly simplifies the complete network design.

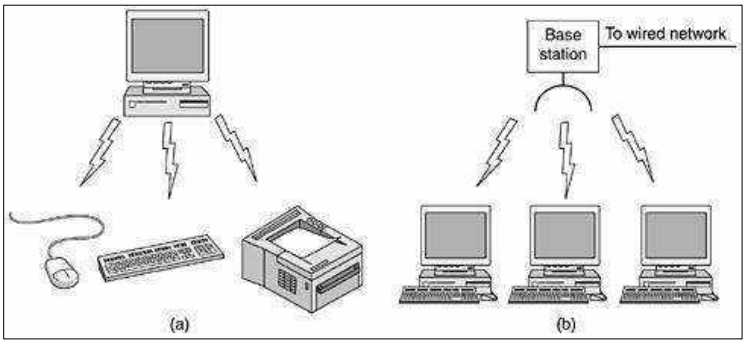
In most wide area networks, the subnet **Relation between hosts on LANs and the subnet** consists of two distinct components: transmission lines and switching elements. Transmission lines move bits between machines. They can be made of copper wire, optical fiber, or even radio links. Switching elements are specialized computers that connect three or more transmission lines. When data arrive on an incoming line, the switching element must choose an outgoing line

on which to forward them. These switching computers have been called by various names in the past; the name router is now most commonly used. Unfortunately, some people pronounce it "rooter" and others have it rhyme with "doubter." Determining the correct pronunciation will be left as an exercise for the reader. (Note: the perceived correct answer may depend on where you live.)

A short comment about the term "subnet" is in order here. Originally, its only meaning was the collection of routers and communication lines that moved packets from the source host to the destination host. However, some years later, it also acquired a second meaning in conjunction with network addressing unfortunately, no widely- used alternative exists for its initial meaning, so with some hesitation we will use it in both senses. From the context, it will always be clear what is meant.

In most WANs, the network contains numerous transmission lines, each one connecting a **pair** of routers. If two routers that do not share a transmission line wish to communicate, they must do this indirectly, via other routers. When a packet is sent from one router to another via one or more intermediate routers, the packet is received at each intermediate router in its entirety, stored there until the required output line is free, and then forwarded. A subnet organized according to this principle is called a store-and-forward or packet-switched subnet. Nearly all wide area networks (except those using satellites) have store-and-forward subnets. When the packets are small and all the same size, they are often called cells.

Wireless Networks



- **Bluetooth configuration.**
- **Wireless LAN.**

Digital wireless communication is not a new idea. As early as 1901, the Italian physicist Guglielmo Marconi demonstrated a ship-to-shore wireless telegraph,

using Morse code (dots and dashes are binary, after all). Modern digital wireless systems have better performance, but the basic idea is the same.

To a first approximation, wireless networks can be divided into three main categories:

- System interconnection.
- Wireless LANs.
- Wireless WANs.

System interconnection is all about interconnecting the components of a computer using short-range radio. Almost every computer has a monitor, keyboard, mouse, and printer connected to the main unit by cables. So many new users have a hard time plugging all the cables into the right little holes (even though they are usually color coded) that most computer vendors offer the option of sending a technician to the user's home to do it. Consequently, some companies got together to design a short-range wireless network called Bluetooth to connect these components without wires. Bluetooth also allows digital cameras, headsets, scanners, and other devices to connect to a computer by merely being brought within range. No cables, no driver installation, just put them down, turn them on, and they work. For many people, this ease of operation is a big plus.

The next step up in wireless networking are the wireless LANs. These are systems in which every computer has a radio, modem and antenna with which it can communicate with other systems. Often there is an antenna on the ceiling that the machines talk to however, if the systems are close enough, they can communicate directly with one another in a peer-to-peer configuration. Wireless LANs are becoming increasingly common in small offices and homes, where installing Ethernet is considered too much trouble, as well as in older office buildings, company cafeterias, conference rooms, and other places. There is a standard for wireless LANs, called IEEE 802.11, which most systems implement and which is becoming very widespread.

The third kind of wireless network is used in wide area systems. The radio network used for cellular telephones is an example of a low-bandwidth wireless system. This system has already gone through three generations. The first generation was analog and for voice only. The second generation was digital and for voice only. The third generation is digital and is for both voice and data. In a certain sense, cellular wireless networks are like wireless LANs, except that the distances involved are much greater and the bit rates much lower. Wireless LANs

can operate at rates up to about 50 Mbps over distances of tens of meters. Cellular systems operate below 1 Mbps, but the distance between the base station and the computer or telephone is measured in kilometers rather than in meters.

In addition to these low-speed networks, high-bandwidth wide area wireless networks are also being developed. The initial focus is high-speed wireless Internet access from homes and businesses, bypassing the telephone system. This service is often called local multipoint distribution service. We will study it later in the book. A standard for it, called IEEE 802.16, has also been developed.

Home Networks

Home networking is on the horizon. The fundamental idea is that in the future most homes will be set up for networking. Every device in the home will be capable of communicating with every other device, and all of them will be accessible over the Internet. This is one of those visionary concepts that nobody asked for (like TV remote controls or mobile phones), but once they arrived nobody can imagine how they lived without them.

Many devices are capable of being networked. Some of the more obvious categories are as follows:

- Computers (desktop PC, notebook PC, PDA, shared peripherals).
- Entertainment (TV, DVD, VCR, camcorder, camera, stereo, MP3).
- Telecommunications (telephone, mobile telephone, intercom, fax).
- Appliances (microwave, refrigerator, clock, furnace, airco, lights).
- Telemetry (utility meter, smoke/burglar alarm, thermostat, baby cam).

Home computer networking is already here in a limited way. Many homes already have a device to connect multiple computers to a fast Internet connection. Networked entertainment is not quite here, but as more and more music and movies can be downloaded from the Internet, there will be a demand to connect stereos and televisions to it. Also, people will want to share their own videos with friends and family, so the connection will need to go both ways. Telecommunications gear is already connected to the outside world, but soon it will be digital and go over the Internet. The average home probably has a dozen clocks (e.g., in appliances), all of which have to be reset twice a year when daylight saving time (summer time) comes and goes. If all the clocks were on the Internet, that resetting could be done automatically. Finally, remote monitoring

of the home and its contents is a likely winner. Probably many parents would be willing to spend some money to monitor their sleeping babies on their PDAs when they are eating out, even with a rented teenager in the house. While one can imagine a separate network for each application area, integrating all of them into a single network is probably a better idea.

Home networking has some fundamentally different properties than other network types. First, the network and devices have to be easy to install. The author has installed numerous pieces of hardware and software on various computers over the years, with mixed results. A series of phone calls to the vendor's helpdesk typically resulted in answers like (1) Read the manual, (2) Reboot the computer, (3) Remove all hardware and software except ours and try again, (4) Download the newest driver from our Web site, and if all else fails, (5) Reformat the hard disk and then reinstall Windows from the CD-ROM. Telling the purchaser of an Internet refrigerator to download and install a new version of the refrigerator's operating system is not going to lead to happy customers. Computer users are accustomed to putting up with products that do not work; the car-, television-, and refrigerator-buying public is far less tolerant. They expect products to work for 100% from the word go.
Second, the network and devices have to be foolproof in operation. Air conditioners used to have one knob with four settings: OFF, LOW, MEDIUM, and HIGH. Now they have 30-page manuals.

Third, low price is essential for success. People will not pay a $50 premium for an Internet thermostat because few people regard monitoring their home temperature from work that important. For $5 extra, it might sell, though.

Fourth, the main application is likely to involve multimedia, so the network needs sufficient capacity. There is no market for Internet-connected televisions that show shaky movies at 320 x 240 pixel resolution and 10 frames/sec. Fast Ethernet, the workhorse in most offices, is not good enough for multimedia. Consequently, home networks will need better performance than that of existing office networks and at lower prices before they become mass market items.

Fifth, it must be possible to start out with one or two devices and expand the reach of the network gradually. This means no format wars. Telling consumers to buy peripherals with IEEE 1394 (FireWire) interfaces and a few years later retracting that and saying USB 2.0 is the interface-of-the-month is

going to make consumers skittish. The network interface will have to remain stable for many years; the wiring (if any) will have to remain stable for decades.

Sixth, security and reliability will be very important. Losing a few files to an e-mail virus is one thing; having a burglar disarm your security system from his PDA and then plunder your house is something quite different.

Network Software

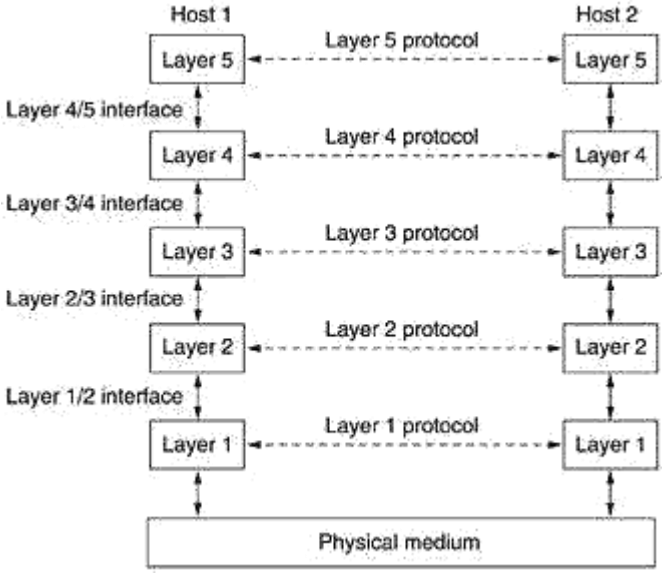
The first computer networks were designed with the hardware as the main concern and the software as an afterthought. This strategy no longer works. Network software is now highly structured. In the following sections we examine the software structuring technique in some detail. The method described here forms the keystone of the entire book and will occur repeatedly later on.

Protocol Hierarchies

To reduce their design complexity, most networks are organized as a stack of layers or levels, each one built upon the one below it. The number of layers, the name of each layer, the contents of each layer, and the function of each layer differ from network to network. The purpose of each layer is to offer certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented. In a sense, each layer is a kind of virtual machine, offering certain services to the layer above it.

This concept is actually a familiar one and used throughout computer science, where it is variously known as information hiding, abstract data types, data encapsulation, and object-oriented programming. The fundamental idea is that a particular piece of software (or hardware) provides a service to its users but keeps the details of its internal state and algorithms hidden from them.

Layers, protocols, and interfaces.



In reality, no data are directly transferred from layer n on one machine to layer n on 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 medium through which actual communication occurs.

Between each pair of adjacent layers is an interface. The interface defines which primitive operations and services the lower layer makes available to the upper one. When network designers decide how many layers to include in a network and what each one should do, one of the most important considerations is defining clean interfaces between the layers. Doing so, in turn, requires that each layer perform a specific collection of well-understood functions. In addition to minimizing the amount of information that must be passed between layers, clear-cut interfaces also make it simpler to replace the implementation of one layer with a completely different implementation (e.g., all the telephone lines are replaced by satellite channels) because all that is required of the new implementation is that it offer exactly the same set of services to its upstairs neighbor as the old implementation did. In fact, it is common that different hosts use different implementations.

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

An analogy may help explain the idea of multilayer communication. Imagine two philosophers (peer processes in layer 3), one of whom speaks Urdu and English and one of whom speaks Chinese and French. Since they have no common language, they each engage a translator (peer processes at layer 2), each of whom in turn contacts a secretary (peer processes in layer 1). Philosopher 1 wishes to convey his affection for oryctolaguscuniculus to his peer. To do so, he passes a message (in English) across the 2/3 interface to his translator, saying "I like rabbits,". The translators have agreed on a neutral language known to both of them, Dutch, so the message is converted to "Ikvindkonijnenleuk." The choice of language is the layer 2 protocol and is up to the layer 2 peer processes.

Design Issues for the Layers

Some of the key design issues that occur in computer networks are present in several layers. Below, we will briefly mention some of the more important ones.

Every layer needs a mechanism for identifying senders and receivers. Since a network normally has many computers, some of which have multiple processes, a means is needed for a process on one machine to specify with whom it wants to talk. As a consequence of having multiple destinations, some form of addressing is needed in order to specify a specific destination.

Another set of design decisions concerns the rules for data transfer. In some systems, data only travel in one direction; in others, data can go both ways. The protocol must also determine how many logical channels the connection

corresponds to and what their priorities are. Many networks provide at least two logical channels per connection, one for normal data and one for urgent data.

Error control is an important issue because physical communication circuits are not perfect. Many error-detecting and error-correcting codes are known, but both ends of the connection must agree on which one is being used. In addition, the receiver must have some way of telling the sender which messages have been correctly received and which have not.

Not all communication channels preserve the order of messages sent on them. To deal with a possible loss of sequencing, the protocol must make explicit provision for the receiver to allow the pieces to be reassembled properly. An obvious solution is to number the pieces, but this solution still leaves open the question of what should be done with pieces that arrive out of order.

An issue that occurs at every level is how to keep a fast sender from swamping a slow receiver with data. Various solutions have been proposed and will be discussed later. Some of them involve some kind of feedback from the receiver to the sender, either directly or indirectly, about the receiver's current situation. Others limit the sender to an agreed-on transmission rate. This subject is called flow control.

Connection-Oriented and Connectionless Services

Layers can offer two different types of service to the layers above them: connection-oriented and connectionless.

In this section we will look at these two types and examine the differences between them.

Connection-oriented service is modeled after the telephone system. To talk to someone, you pick up the phone, dial the number, talk, and then hang up. Similarly, to use a connection-oriented network service, the service user first establishes a connection, uses the connection, and then releases the connection. The essential aspect of a connection is that it acts like a tube: the sender pushes objects (bits) in at one end, and the receiver takes them out at the other end. In most cases the order is preserved so that the bits arrive in the order they were sent.

In some cases when a connection is established, the sender, receiver, and subnet conduct a negotiation about parameters to be used, such as maximum message size, quality of service required, and other issues. Typically, one side

makes a proposal and the other side can accept it, reject it, or make a counterproposal.

In contrast, connectionless service is modeled after the postal system. Each message (letter) carries the full destination address, and each one is routed through the system independent of all the others. Normally, when two messages are sent to the same destination, the first one sent will be the first one to arrive. However, it is possible that the first one sent can be delayed so that the second one arrives first.

Each service can be characterized by a quality of service. Some services are reliable in the sense that they never lose data. Usually, a reliable service is implemented by having the receiver acknowledge the receipt of each message so the sender is sure that it arrived. The acknowledgement process introduces overhead and delays, which are often worth it but are sometimes undesirable.

Six different types of service.

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Remote login
	Unreliable connection	Digitized voice
Connection-less	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Registered mail
	Request-reply	Database query

A typical situation in which a reliable connection-oriented service is appropriate is file transfer. The owner of the file wants to be sure that all the bits arrive correctly and in the same order they were sent. Very few file transfer customers would prefer a service that occasionally scrambles or loses a few bits, even if it is much faster.

Service Primitives Five service primitives for implementing a simple connection-oriented service

Primitive	Meaning
LISTEN	Block waiting for an incoming connection
CONNECT	Establish a connection with a waiting peer
RECEIVE	Block waiting for an incoming message
SEND	Send a message to the peer
DISCONNECT	Terminate a connection

A service is formally specified by a set of primitives (operations) available to a user process to access the service. These primitives tell the service to perform some action or report on an action taken by a peer entity. If the protocol stack is located in the operating system, as it often is, the primitives are normally system calls. These calls cause a trap to kernel mode, which then turns control of the machine over to the operating system to send the necessary packets.

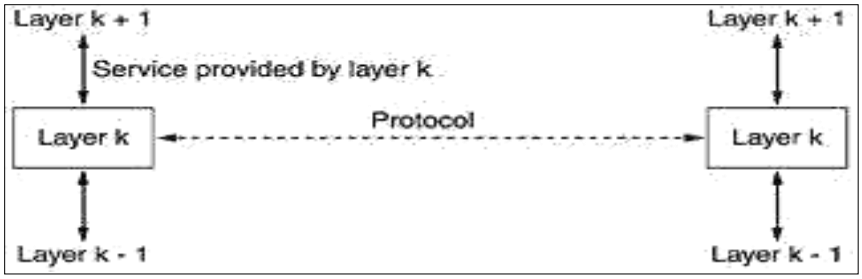
The set of primitives available depends on the nature of the service being provided. The primitives for connection-oriented service are different from those of connectionless service. As a minimal example of the service primitives that might be provided to implement a reliable byte stream in a client-server environment**.**

These primitives might be used as follows. First, the server executes LISTEN to indicate that it is prepared to accept incoming connections. A common way to implement LISTEN is to make it a blocking system call. After executing the primitive, the server process is blocked until a request for connection appears.

The Relationship of Services to Protocols

Services and protocols are distinct concepts, although they are frequently confused. This distinction is so important, however, that we emphasize it again here. A service is a set of primitives (operations) that a layer provides to the layer above it. The service defines what operations the layer is prepared to perform on behalf of its users, but it says nothing at all about how these operations are implemented. A service relates to an interface between two layers, with the lower layer being the service provider and the upper layer being the service user.

The relationship between a service and a protocol



A protocol, in contrast, is a set of rules governing the format and meaning of the packets, or messages that are exchanged by the peer entities within a layer. Entities use protocols to implement their service definitions. They are free to change their protocols at will, provided they do not change the service visible to their users. In this way, the service and the protocol are completely decoupled.

In other words, services relate to the interfaces between layers, In contrast, protocols relate to the packets sent between peer entities on different machines. It is important not to confuse the two concepts.

An analogy with programming languages is worth making. A service is like an abstract data type or an object in an object-oriented language. It defines operations that can be performed on an object but does not specify how these operations are implemented. A protocol relates to the implementation of the service and as such is not visible to the user of the service.

Many older protocols did not distinguish the service from the protocol. In effect, a typical layer might have had a service primitive SEND PACKET with the user providing a pointer to a fully assembled packet. This arrangement meant that all changes to the protocol were immediately visible to the users. Most network designers now regard such a design as a serious blunder.

Reference Models

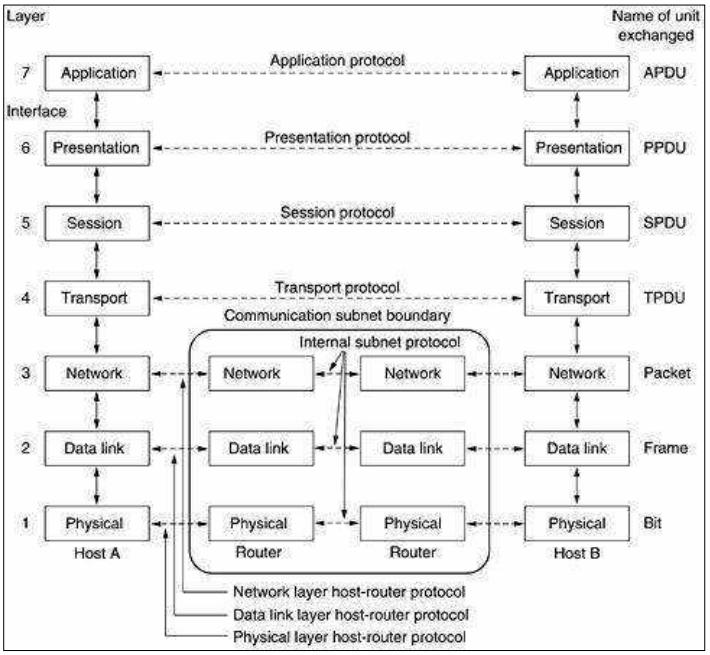
Two important network architectures, the OSI reference model and the TCP/IP reference model. Although the protocols associated with the OSI model are rarely used any more, the model itself is actually quite general and still valid, and the features discussed at each layer are still very important. The TCP/IP model has the opposite properties: the model itself is not of much use but the

protocols are widely used. For this reason we will look at both of them in detail. Also, sometimes you can learn more from failures than from successes.

The OSI Reference Model

The OSI model (minus the physical medium) is shown in Fig. 1-20. This model is based on a proposal developed by the International Standards Organization (ISO) as a first step toward international standardization of the protocols used in the various layers (Day and Zimmermann, 1983). It was revised in 1995 (Day, 1995). The model is called the ISO OSI (Open Systems Interconnection) Reference Model because it deals with connecting open systems- that is, systems that are open for communication with other systems. We will just call it the OSI model for short.

The OSI reference model.



The OSI model has seven layers. The principles that were applied to arrive at the seven layers can be briefly summarized as follows:

- A layer should be created where a different abstraction is needed.
- Each layer should perform a well-defined function.
- The function of each layer should be chosen with an eye toward defining internationally standardized protocols.

- The layer boundaries should be chosen to minimize the information flow across the interfaces.
- The number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity and small enough that the architecture does not become unwieldy.

Below we will discuss each layer of the model in turn, starting at the bottom layer. Note that the OSI model itself is not a network architecture because it does not specify the exact services and protocols to be used in each layer. It just tells what each layer should do. However, ISO has also produced standards for all the layers, although these are not part of the reference model itself. Each one has been published as a separate international standard.

The Physical Layer

The physical layer is concerned with transmitting raw bits over a communication channel. The design issues have to do with making sure that when one side sends a 1 bit, it is received by the other side as a 1 bit, not as a 0 bit. Typical questions here are how many volts should be used to represent a 1 and how many for a 0, how many nanoseconds a bit lasts, whether transmission may proceed simultaneously in both directions, how the initial connection is established and how it is torn down when both sides are finished, and how many pins the network connector has and what each pin is used for. The design issues here largely deal with mechanical, electrical, and timing interfaces, and the physical transmission medium, which lies below the physical layer.

The Data Link Layer

The main task of the data link layer is to transform a raw transmission facility into a line that appears free of undetected transmission errors to the network layer. It accomplishes this task by having the sender break up the input data into data frames (typically a few hundred or a few thousand bytes) and transmit the frames sequentially. If the service is reliable, the receiver confirms correct receipt of each frame by sending back an acknowledgement frame.

Another issue that arises in the data link layer (and most of the higher layers as well) is how to keep a fast transmitter from drowning a slow receiver in data. Some traffic regulation mechanism is often needed to let the transmitter know how much buffer space the receiver has at the moment. Frequently, this flow regulation and the error handling are integrated.

Broadcast networks have an additional issue in the data link layer: how to control access to the shared channel.
A special sub-layer of the data link layer, the medium access control sub-layer, deals with this problem.

The Network Layer

The network layer controls the operation of the subnet. A key design issue is determining how packets are routed from source to destination. Routes can be based on static tables that are "wired into" the network and rarely changed. They can also be determined at the start of each conversation, for example, a terminal session (e.g., a login to a remote machine). Finally, they can be highly dynamic, being determined anew for each packet, to reflect the current network load.

If too many packets are present in the subnet at the same time, they will get in one another's way, forming bottlenecks. The control of such congestion also belongs to the network layer. More generally, the quality of service provided (delay, transit time, jitter, etc.) is also a network layer issue.

When a packet has to travel from one network to another to get to its destination, many problems can arise. The addressing used by the second network may be different from the first one. The second one may not accept the packet at all because it is too large. The protocols may differ, and so on. It is up to the network layer to overcome all these problems to allow heterogeneous networks to be interconnected.

The Transport Layer

The basic function of the transport layer is to accept data from above, split it up into smaller units if need be, pass these to the network layer, and ensure that the pieces all arrive correctly at the other end. Furthermore, all this must be done efficiently and in a way that isolates the upper layers from the inevitable changes in the hardware technology.

The transport layer also determines what type of service to provide to the session layer, and, ultimately, to the users of the network. The most popular type of transport connection is an error-free point-to-point channel that delivers messages or bytes in the order in which they were sent. However, other possible kinds of transport service are the transporting of isolated messages, with no guarantee about the order of delivery, and the broadcasting of messages to multiple destinations. The type of service is determined when the connection is

established. (As an aside, an error-free channel is impossible to achieve; what people really mean by this term is that the error rate is low enough to ignore in practice.)

The transport layer is a true end-to-end layer, all the way from the source to the destination. In other words, a program on the source machine carries on a conversation with a similar program on the destination machine, using the message headers and control messages. In the lower layers, the protocols are between each machine and its immediate neighbors, and not between the ultimate source and destination machines, which may be separated by many routers. The difference between layers 1 through 3, which are chained, and layers 4 through 7, which are end-to-end,.

The Session Layer

The session layer allows users on different machines to establish sessions between them. Sessions offer various services, including dialog control (keeping track of whose turn it is to transmit), token management (preventing two parties from attempting the same critical operation at the same time), and synchronization (check pointing long transmissions to allow them to continue from where they were after a crash).

The Presentation Layer

Unlike lower layers, which are mostly concerned with moving bits around, the presentation layer is concerned with the syntax and semantics of the information transmitted. In order to make it possible for computers with different data representations to communicate, the data structures to be exchanged can be defined in an abstract way, along with a standard encoding to be used "on the wire." The presentation layer manages these abstract data structures and allows higher-level data structures (e.g., banking records), to be defined and exchanged.

The Application Layer

The application layer contains a variety of protocols that are commonly needed by users. One widely-used application protocol is HTTP (HyperText Transfer Protocol), which is the basis for the World Wide Web. When a browser wants a Web page, it sends the name of the page it wants to the server using HTTP. The server then sends the page back. Other application protocols are used for file transfer, electronic mail, and network news.

The TCP/IP Reference Model

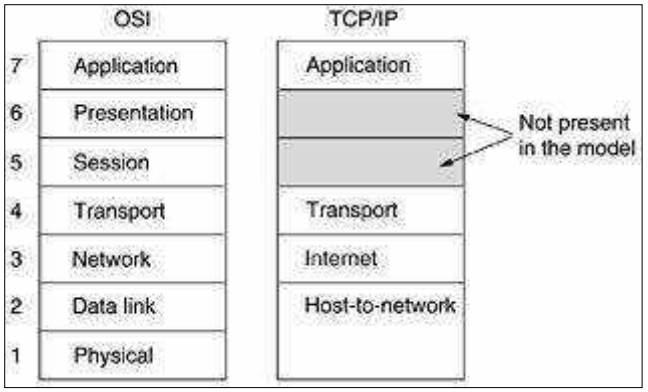
Let us now turn from the OSI reference model to the reference model used in the grandparent of all wide area computer networks, the ARPANET, and its successor, the worldwide Internet. Although we will give a brief history of the ARPANET later, it is useful to mention a few key aspects of it now. The ARPANET was a research network sponsored by the DoD (U.S. Department of Defense). It eventually connected hundreds of universities and government installations, using leased telephone lines. When satellite and radio networks were added later, the existing protocols had trouble interworking with them, so a new reference architecture was needed. Thus, the ability to connect multiple networks in a seamless way was one of the major design goals from the very beginning. This architecture later became known as the TCP/IP Reference Model, after its two primary protocols. It was first defined in (Cerf and Kahn, 1974). A later perspective is given in (Leiner et al., 1985). The design philosophy behind the model is discussed in (Clark, 1988).

Given the DoD's worry that some of its precious hosts, routers, and internetwork gateways might get blown topieces at a moment's notice, another major goal was that the network be able to survive loss of subnet hardware, with existing conversations not being broken off. In other words, DoD wanted connections to remain intact as long as the source and destination machines were functioning, even if some of the machines or transmission lines in between were suddenly put out of operation. Furthermore, a flexible architecture was needed since applications with divergent requirements were envisioned, ranging from transferring files to real-time speech transmission.

The Internet Layer

All these requirements led to the choice of a packet-switching network based on a connectionless internetwork layer. This layer, called the internet layer, is the linchpin that holds the whole architecture together. Its job is to permit hosts to inject packets into any network and have them travel independently to the destination (potentially on a different network). They may even arrive in a different order than they were sent, in which case it is the job of higher layers to rearrange them, if in-order delivery is desired. Note that "internet" is used here in a generic sense, even though this layer is present in the Internet.

The TCP/IP reference model

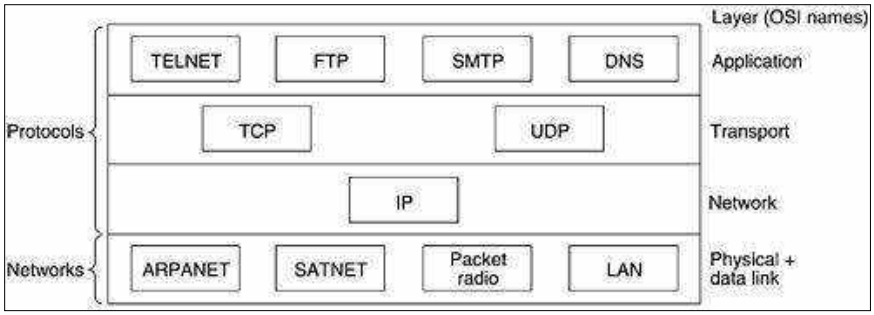


The analogy here is with the (snail) mail system. A person can drop a sequence of international letters into a mail box in one country, and with a little luck, most **of** them will be delivered to the correct address in the destination country. Probably the letters will travel through one or more international mail gateways along the way, but this is transparent to the users. Furthermore, that each country (i.e., each network) has its own stamps, preferred envelope sizes, and delivery rules is hidden from the users.

The Transport Layer

The layer above the internet layer in the TCP/IP model is now usually called the transport layer. It is designed to allow peer entities on the source and destination hosts to carry on a conversation, just as in the OSI transport layer. Two end-to-end transport protocols have been defined here. The first one, TCP (Transmission Control Protocol), is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error on any other machine in the internet. It fragments the incoming byte stream into discrete messages and passes each one on to the internet layer. At the destination, the receiving TCP process reassembles the received messages into the output stream. TCP also handles flow control to make sure a fast sender cannot swamp a slow receiver with more messages than it can handle.

Protocols and networks in the TCP/IP model initially.



The second protocol in this layer, UDP (User Datagram Protocol), is an unreliable, connectionless protocol for applications that do not want TCP's sequencing or flow control and wish to provide their own. It is also widely used for one-shot, client-server-type request-reply queries and applications in which prompt delivery is more important than accurate delivery, such as transmitting speech or video. The relation of IP, TCP, and UDP is shown in Fig. 1-22. Since the model was developed, IP has been implemented on many other networks.

The Application Layer

The TCP/IP model does not have session or presentation layers. No need for them was perceived, so they were not included. Experience with the OSI model has proven this view correct: they are of little use to most applications.

On top of the transport layer is the application layer. It contains all the higher-level protocols. The early ones included virtual terminal (TELNET), file transfer (FTP), and electronic mail (SMTP). The virtual terminal protocol allows a user on one machine to log onto a distant machine and work there. The file transfer protocol provides a way to move data efficiently from one machine to another. Electronic mail was originally just a kind of file transfer, but later a specialized protocol (SMTP) was developed for it. Many other protocols have been added to these over the years: the Domain Name System (DNS) for mapping host names onto their network addresses, NNTP, the protocol for moving USENET news articles around, and HTTP, the protocol for fetching pages on the World Wide Web, and many others.

The Host-to-Network Layer

Below the internet layer is a great void. The TCP/IP reference model does not really say much about what happens here, except to point out that the host has

to connect to the network using some protocol so it can send IP packets to it. This protocol is not defined and varies from host to host and network to network.

A Comparison of the OSI and TCP/IP Reference Models

The OSI and TCP/IP reference models have much in common. Both are based on the concept of a stack of independent protocols. Also, the functionality of the layers is roughly similar. For example, in both models the layers up through and including the transport layer are there to provide an end-to-end, network-independent transport service to processes wishing to communicate. These layers form the transport provider. Again in both models, the layers above transport are application-oriented users of the transport service.

Despite these fundamental similarities, the two models also have many differences. In this section we will focus on the key differences between the two reference models. It is important to note that we are comparing the reference models here, not the corresponding protocol stacks. The protocols themselves will be discussed later.

Three concepts are central to the OSI model:

- Services.
- Interfaces.
- Protocols.

Probably the biggest contribution of the OSI model is to make the distinction between these three concepts explicit. Each layer performs some services for the layer above it. The service definition tells what the layer does, not how entities above it access it or how the layer works. It defines the layer's semantics.

A layer's interface tells the processes above it how to access it. It specifies what the parameters are and what results to expect. It, too, says nothing about how the layer works inside.

Finally, the peer protocols used in a layer are the layer's own business. It can use any protocols it wants to, as long as it gets the job done (i.e., provides the offered services). It can also change them at will without affecting software in higher layers.

These ideas fit very nicely with modern ideas about object-oriented programming. An object, like a layer, has a set of methods (operations) that

processes outside the object can invoke. The semantics of these methods define the set of services that the object offers. The methods' parameters and results from the object's interface. The code internal to the object is its protocol and is not visible or of any concern outside the object.

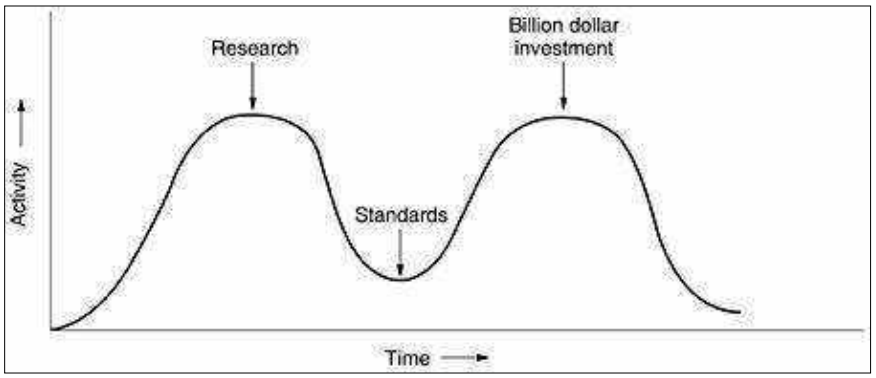
The TCP/IP model did not originally clearly distinguish between service, interface, and protocol, although people have tried to retrofit it after the fact to make it more OSI-like. For example, the only real services offered by the internet layer are SEND IP PACKET and RECEIVE IP PACKET.

A Critique of the OSI Model and Protocols

Neither the OSI model and its protocols nor the TCP/IP model and its protocols are perfect. Quite a bit of criticism can be, and has been, directed at both of them. In this section and the next one, we will look at some of these criticisms. We will begin with OSI and examine TCP/IP afterward.

At the time the second edition of this book was published (1989), it appeared to many experts in the field that the OSI model and its protocols were going to take over the world and push everything else out of their way. This did not happen. These lessons can be summarized as:

The apocalypse of the two elephants



- Bad timing.
- Bad technology.
- Bad implementations.
- Bad politics.

1. Bad Timing

First let us look at reason one: bad timing. The time at which a standard is established is absolutely critical to its success. David Clark of M.I.T. has a theory of standards that he calls the apocalypse of the two elephants.

2. Bad Technology

The second reason that OSI never caught on is that both the model and the protocols are flawed. The choice of seven layers was more political than technical, and two of the layers (session and presentation) are nearly empty, whereas two other ones (data link and network) are overfull.

3. Bad Implementations

Given the enormous complexity of the model and the protocols, it will come as no surprise that the initial implementations were huge, unwieldy, and slow. Everyone who tried them got burned. It did not take long for people to associate "OSI" with "poor quality." Although the products improved in the course of time, the image stuck.

4. Bad Politics

On account of the initial implementation, many people, especially in academia, thought of TCP/IP as part of UNIX, and UNIX in the 1980s in academia was not unlike parenthood (then incorrectly called motherhood) and apple pie.

A Critique of the TCP/IP Reference Model

The TCP/IP model and protocols have their problems too. First, the model does not clearly distinguish the concepts of service, interface, and protocol. Good software engineering practice requires differentiating between the specification and the implementation, something that OSI does very carefully, and TCP/IP does not. Consequently, the TCP/IP model is not much of a guide for designing new networks using new technologies.

Second, the TCP/IP model is not at all general and is poorly suited to describing any protocol stack other than TCP/IP. Trying to use the TCP/IP model to describe Bluetooth, for example, is completely impossible.

Third, the host-to-network layer is not really a layer at all in the normal sense of the term as used in the context of layered protocols. It is an interface (between the network and data link layers). The distinction between an interface and a layer is crucial, and one should not be sloppy about it.

Fourth, the TCP/IP model does not distinguish (or even mention) the physical and data link layers. These are completely different. The physical layer has to do with the transmission characteristics of copper wire, fiber optics, and wireless communication. The data link layer's job is to delimit the start and end of frames and get them from one side to the other with the desired degree of reliability. A proper model should include both as separate layers. The TCP/IP model does not do this.

The hybrid reference model to be used

Finally, although the IP and TCP protocols were carefully thought out and well implemented, many of the other protocols were ad hoc, generally produced by a couple of graduate students hacking away until they got tired. The protocol implementations were then distributed free, which resulted in their becoming widely used, deeply entrenched, and thus hard to replace. Some of them are a bit of an embarrassment now.

UNIT – II

PHYSICAL LAYER

Guided Transmission Media

The purpose of the physical layer is to transport a raw bit stream from one machine to another. Various physical media can be used for the actual transmission. Each one has its own niche in terms of bandwidth, delay, cost, and ease of installation and maintenance. Media are roughly grouped into guided media, such as copper wire and fiber optics, and unguided media, such as radio and lasers through the air. We will look at all of these in the following sections.

Magnetic Media

One of the most common ways to transport data from one computer to another is to write them onto magnetic tape or removable media (e.g., recordable DVDs), physically transport the tape or disks to the destination machine, and read them back in again. Although this method is not as sophisticated as using a geosynchronous communication satellite, it is often more cost effective, especially for applications in which high bandwidth or cost per bit transported is the key factor.

A simple calculation will make this point clear. An industry standard Ultrium tape can hold 200 gigabytes. A box 60 x 60 x 60 cm can hold about 1000 of these tapes, for a total capacity of 200 terabytes, or 1600 terabits (1.6 petabits). A box of tapes can be delivered anywhere in the United States in 24 hours by Federal Express and other companies. The effective bandwidth of this transmission is 1600 terabits/86,400 sec, or 19 Gbps. If the destination is only an hour away by road, the bandwidth is increased to over 400 Gbps. No computer network can even approach this.

For a bank with many gigabytes of data to be backed up daily on a second machine (so the bank can continue to function even in the face of a major flood or earthquake), it is likely that no other transmission technology can even begin to approach magnetic tape for performance. Of course, networks are getting faster, but tape densities are increasing, too.

Twisted Pair

Although the bandwidth characteristics of magnetic tape are excellent, the delay characteristics are poor. Transmission time is measured in minutes or hours, not milliseconds. For many applications an on-line connection is needed. One of the oldest and still most common transmission media is twisted pair. A twisted pair consists of two insulated copper wires, typically about 1 mm thick. The wires are twisted together in a helical form, just like a DNA molecule. Twisting is done because two parallel wires constitute a fine antenna. When the wires are twisted, the waves from different twists cancel out, so the wire radiates less effectively.

Category **3 UTP. (b) Category 5 UTP.**

The most common application of the twisted pair is the telephone system. Nearly all telephones are connected to the telephone company (telco) office by a twisted pair. Twisted pairs can run several kilometers without amplification, but for longer distances, repeaters are needed. When many twisted pairs run in parallel for a substantial distance, such as all the wires coming from an apartment building to the telephone company office, they are bundled together and encased in a protective sheath. The pairs in these bundles would interfere with one another if it were not for the twisting. In parts of the world where telephone lines run on poles above ground, it is common to see bundles several centimeters in diameter.

Twisted pairs can be used for transmitting either analog or digital signals. The bandwidth depends on the thickness of the wire and the distance traveled, but several megabits/sec can be achieved for a few kilometers in many cases. Due to their adequate performance and low cost, twisted pairs are widely used and are likely to remain so for years to come.

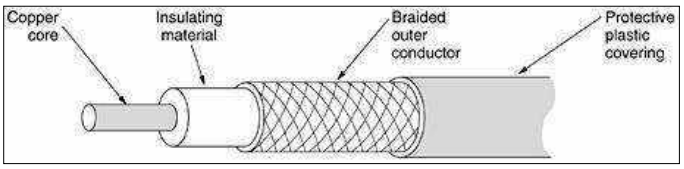
Twisted pair cabling comes in several varieties, two of which are important for computer networks. Category 3 twisted pairs consist of two insulated wires gently twisted together. Four such pairs are typically grouped in a plastic sheath to protect the wires and keep them together. Prior to about 1988, most office buildings had one category 3 cable running from a central wiring closet on each floor into each office. This scheme allowed up to four regular telephones or two

multiline telephones in each office to connect to the telephone company equipment in the wiring closet.

Coaxial Cable

Another common transmission medium is the coaxial cable (known to its many friends as just "coax" and pronounced "co-ax"). It has better shielding than twisted pairs, so it can span longer distances at higher speeds. Two kinds of coaxial cable are widely used. One kind, 50-ohm cable, is commonly used when it is intended for digital transmission from the start. The other kind, 75-ohm cable, is commonly used for analog transmission and cable television but is becoming more important with the advent of Internet over cable. This distinction is based on historical, rather than technical, factors (e.g., early dipole antennas had an impedance of 300 ohms, and it was easy to use existing 4:1 impedance matching transformers).

A coaxial cable

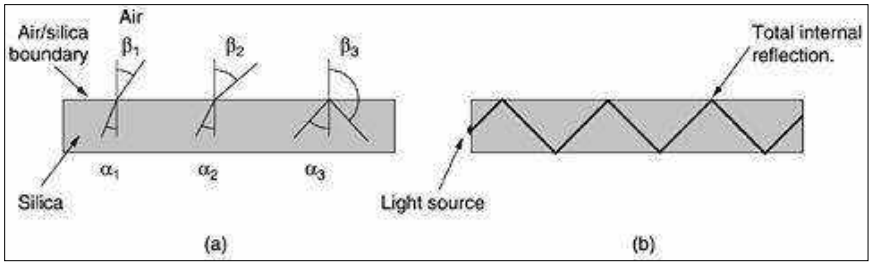


A coaxial cable consists of a stiff copper wire as the core, surrounded by an insulating material. The insulator is encased by a cylindrical conductor, often as a closely-woven braided mesh. The outer conductor is covered in a protective plastic sheath.

The construction and shielding of the coaxial cable give it a good combination of high bandwidth and excellent noise immunity. The bandwidth possible depends on the cable quality, length, and signal-to-noise ratio of the data signal. Modern cables have a bandwidth of close to 1 GHz. Coaxial cables used to be widely used within the telephone system for long-distance lines but have now largely been replaced by fiber optics on long-haul routes. Coax is still widely used for cable television and metropolitan area networks, however.

(a) Three examples of a light ray from inside a silica fiber impinging on the air/silica boundary at different angles.
(b) Light trapped by total internal reflection.

Fiber Optics



Many people in the computer industry take enormous pride in how fast computer technology is improving. The original (1981) IBM PC ran at a clock speed of 4.77 MHz. Twenty years later, PCs could run at 2 GHz, a gain of a factor of 20 per decade. Not too bad.

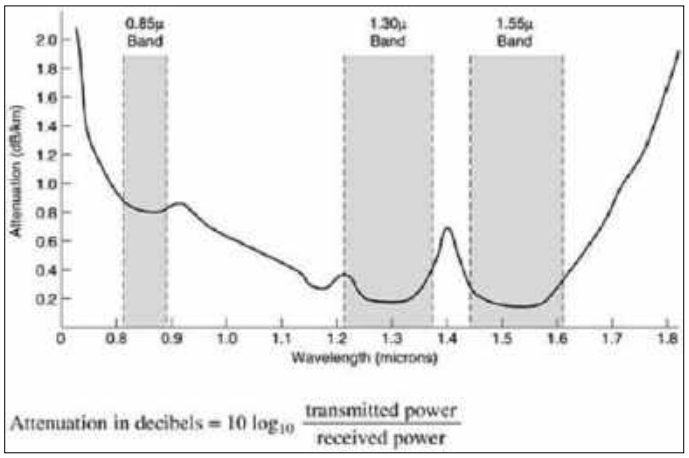
In the same period, wide area data communication went from 56 kbps (the ARPANET) to 1 Gbps (modern optical communication), a gain of more than a factor of 125 per decade, while at the same time the error rate went from 10^{-5} per bit to almost zero.

Furthermore, single CPUs are beginning to approach physical limits, such as speed of light and heat dissipation problems. In contrast, with current fiber technology, the achievable bandwidth is certainly in excess of 50,000 Gbps (50 Tbps) and many people are looking very hard for better technologies and materials. The current practical signaling limit of about 10 Gbps is due to our inability to convert between electrical and optical signals any faster, although in the laboratory, 100 Gbps has been achieved on a single fiber.

In the race between computing and communication, communication won. The full implications of essentially infinite bandwidth (although not at zero cost) have not yet sunk in to a generation of computer scientists and engineers taught to think in terms of the low Nyquist and Shannon limits imposed by copper wire. The new conventional wisdom should be that all computers are hopelessly slow and that networks should try to avoid computation at all costs, no matter how much bandwidth that wastes. In this section we will study fiber optics to see how that transmission technology works.

Transmission of Light through Fiber

Attenuation of light through fiber in the infrared region



Optical fibers are made of glass, which, in turn, is made from sand, an inexpensive raw material available in unlimited amounts. Glassmaking was known to the ancient Egyptians, but their glass had to be no more than 1thick or the light could not shine through. Glass transparent enough to be useful for windows was developed during the Renaissance. The glass used for modern optical fibers is so transparent that if the oceans were full of it instead of water, the seabed would be as visible from the surface as the ground is from an airplane on a clear day.

The attenuation of light through glass depends on the wavelength of the light (as well as on some physical properties of the glass). For the kind of glass used in fibers, decibels per linear kilometer of fiber. The attenuation in decibels is given by the formula.

Fiber Cables

Fiber optic cables are similar to coax, except without the braid. Shows a single fiber viewed from the side. At the center is the glass core through which the light propagates. In multimode fibers, the core is typically 50 microns in diameter, about the thickness of a human hair. In single-mode fibers, the core is 8 to 10 microns.

The core is surrounded by a glass cladding with a lower index of refraction than the core, to keep all the light in the core. Next comes a thin plastic jacket to protect the cladding. Fibers are typically grouped in bundles, protected by an outer sheath. Terrestrial fiber sheaths are normally laid in the ground within a meter of the surface, where they are occasionally subject to attacks by backhoes or gophers. Near the shore, transoceanic fiber sheaths are buried in trenches by a

kind of seaplow. In deep water, they just lie on the bottom, where they can be snagged by fishing trawlers or attacked by giant squid.

Fibers can be connected in three different ways.

First, they can terminate in connectors and be plugged into fiber sockets. Connectors lose about 10 to 20 percent of the light, but they make it easy to reconfigure systems.

Second, they can be spliced mechanically. Mechanical splices just lay the two carefully-cut ends next to each other in a special sleeve and clamp them in place. Alignment can be improved by passing light through the junction and then making small adjustments to maximize the signal. Mechanical splices take trained personnel about 5 minutes and result in a 10 percent light loss.

Third, two pieces of fiber can be fused (melted) to form a solid connection. A fusion splice is almost as good as a single drawn fiber, but even here, a small amount of attenuation occurs.

Wireless Transmission

Our age has given rise to information junkies: people who need to be on-line all the time. For these mobile users, twisted pair, coax, and fiber optics are of no use. They need to get their hits of data for their laptop, notebook, shirt pocket, palmtop, or wristwatch computers without being tethered to the terrestrial communication infrastructure. For these users, wireless communication is the answer. In the following sections, we will look at wireless communication in general, as it has many other important applications besides providing connectivity to users who want to surf the Web from the beach.

Some people believe that the future holds only two kinds of communication: fiber and wireless. All fixed (i.e., non mobile) computers, telephones, faxes, and so on will use fiber, and all mobile ones will use wireless.

Wireless has advantages for even fixed devices in some circumstances. For example, if running a fiber to a building is difficult due to the terrain (mountains, jungles, swamps, etc.), wireless may be better. It is noteworthy that modern wireless digital communication began in the Hawaiian Islands, where large chunks of Pacific Ocean separated the users and the telephone system was inadequate.

The Electromagnetic Spectrum

When electrons move, they create electromagnetic waves that can propagate through space (even in a vacuum). These waves were predicted by the British physicist James Clerk Maxwell in 1865 and first observed by the German physicist Heinrich Hertz in 1887. The number of oscillations per second of a wave is called its frequency, f, and is measured in Hz (in honor of Heinrich Hertz). The distance between two consecutive maxima (or minima) is called the wavelength, which is universally designated by the Greek letter l (lambda).

When an antenna of the appropriate size is attached to an electrical circuit, the electromagnetic waves can be broadcast efficiently and received by a receiver some distance away. All wireless communication is based on this principle.

In vacuum, all electromagnetic waves travel at the same speed, no matter what their frequency. This speed, usually called the speed of light, c, is approximately 3 x 10^8 m/sec, or about 1 foot (30 cm) per nanosecond. (A case could be made for redefining the foot as the distance light travels in a vacuum in 1 nsec rather than basing it on the shoe size of some long-dead king.) In copper or fiber the speed slows to about 2/3 of this value and becomes slightly frequency dependent. The speed of light is the ultimate speed limit. No object or signal can ever move faster than it.

The fundamental relation between f, l, and c (in vacuum) is

Equation 1

Since c is a constant, if we know f, we can find l, and vice versa. As a rule of thumb, when l is in meters and f is

in MHz, lf 300. For example, 100-MHz waves are about 3 meters long, 1000-MHz waves are 0.3-meters long, and 0.1-meter waves have a frequency of 3000 MHz.

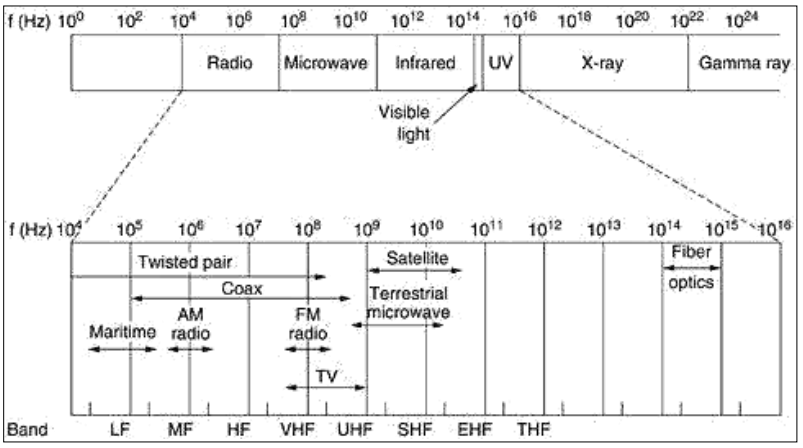
The amount of information that an electromagnetic wave can carry is related to its bandwidth. With current technology, it is possible to encode a few bits per Hertz at low frequencies, but often as many as 8 at high frequencies, so a coaxial cable with a 750 MHz bandwidth can carry several gigabits/sec.

If we solve for f and differentiate with respect to l, we get $\frac{df}{d\lambda} = -\frac{c}{\lambda^2}$
If we now go to finite differences instead of differentials and only look at absolute values, we get $\Delta f = \frac{c\Delta\lambda}{\lambda^2}$

Equation 2

Thus, given the width of a wavelength band, Dl, we can compute the corresponding frequency band, Df, and from that the data rate the band can produce. The wider the band, the higher the data rate. As an example, consider the 1.30-micron band of here we have l=1.3 x 10^{-6} and Dl = 0.17 x 10^{-6},soDf is about 30 THz. At, say, 8 bits/Hz, we get 240 Tbps.

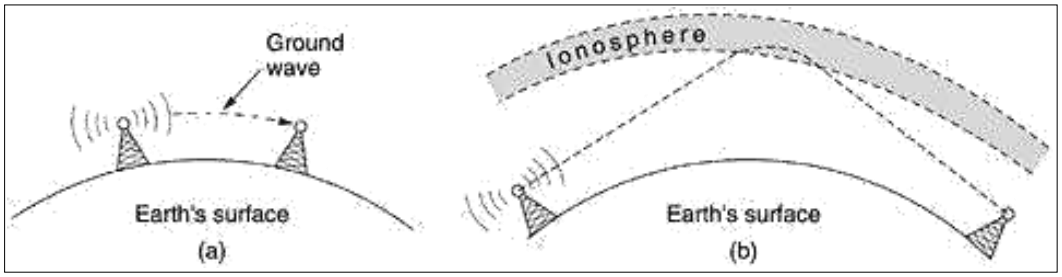
The electromagnetic spectrum and its uses for communication



Most transmissions use a narrow frequency band (i.e., Df/f≪1) to get the best reception (many watts/Hz). However, in some cases, a wide band is used, with two variations. In frequency hopping spread spectrum, the transmitter hops from frequency to frequency hundreds of times per second. It is popular for military communication because it makes transmissions hard to detect and next to impossible to jam. It also offers good resistance to multipath fading because the direct signal always arrives at the receiver first. Reflected signals follow a longer path and arrive later. By then the receiver may have changed frequency and no longer accepts signals on the previous frequency, thus eliminating interference between the direct and reflected signals. In recent years, this technique has also been applied commercially—both 802.11 and Bluetooth use it.

2.3.2 Radio Transmission

(a) In the VLF, LF, and MF bands, radio waves follow the curvature of the earth. (b) In the HF band, they bounce off the ionosphere.



Radio waves are easy to generate, can travel long distances, and can penetrate buildings easily, so they are widely used for communication, both indoors and outdoors. Radio waves also are Omni directional, meaning that they travel in all directions from the source, so the transmitter and receiver do not have to be carefully aligned physically.

Sometimes Omni directional radio is good, but sometimes it is bad. In the 1970s, General Motors decided to equip all its new Cadillacs with computer-controlled antilock brakes. When the driver stepped on the brake pedal, the computer pulsed the brakes on and off instead of locking them on hard. One fine day an Ohio Highway Patrolman began using his new mobile radio to call headquarters, and suddenly the Cadillac next to him began behaving like a bucking bronco. When the officer pulled the car over, the driver claimed that he had done nothing and that the car had gone crazy.

Eventually, a pattern began to emerge: Cadillacs would sometimes go berserk, but only on major highways in Ohio and then only when the Highway Patrol was watching. For a long, long time General Motors could not understand why Cadillacs worked fine in all the other states and also on minor roads in Ohio. Only after much searching did they discover that the Cadillac's wiring made a fine antenna for the frequency used by the Ohio Highway Patrol's new radio system.

The properties of radio waves are frequency dependent. At low frequencies, radio waves pass through obstacles well, but the power falls off sharply with distance from the source, roughly as $1/r^2$ in air. At high frequencies, radio waves tend to travel in straight lines and bounce off obstacles. They are also absorbed by rain. At all frequencies, radio waves are subject to interference from motors and other electrical equipment.

Due to radio's ability to travel long distances, interference between users is a problem. For this reason, all governments tightly license the use of radio transmitters, with one exception, discussed below.

In the VLF, LF, and MF bands, radio waves follow the ground, these waves can be detected for perhaps 1000 km at the lower frequencies, less at the higher

ones. AM radio broadcasting uses the MF band, which is why the ground waves from Boston AM radio stations cannot be heard easily in New York. Radio waves in these bands pass through buildings easily, which is why portable radios work indoors. The main problem with using these bands for data communication is their low bandwidth

In the HF and VHF bands, the ground waves tend to be absorbed by the earth. However, the waves that reach the ionosphere, a layer of charged particles circling the earth at a height of 100 to 500 km, are refracted by it and sent back to earth,. Under certain atmospheric conditions, the signals can bounce several times. Amateur radio operators (hams) use these bands to talk long distance. The military also communicate in the HF and VHF bands.

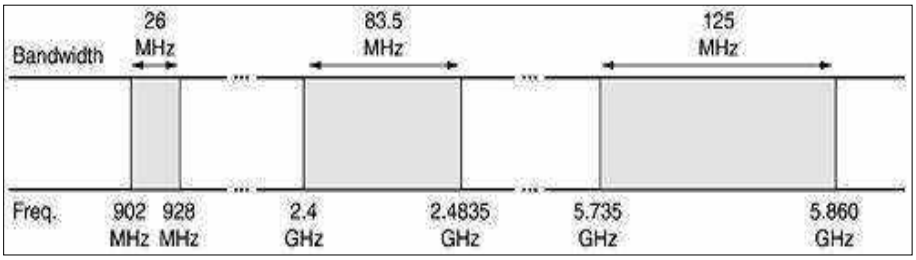
Microwave Transmission

Above 100 MHz, the waves travel in nearly straight lines and can therefore be narrowly focused. Concentrating all the energy into a small beam by means of a parabolic antenna (like the familiar satellite TV dish) gives a much higher signal-to-noise ratio, but the transmitting and receiving antennas must be accurately aligned with each other. In addition, this directionality allows multiple transmitters lined up in a row to communicate with multiple receivers in a row without interference, provided some minimum spacing rules are observed. Before fiber optics, for decades these microwaves formed the heart of the long-distance telephone transmission system. In fact, MCI, one of AT&T's first competitors after it was deregulated, built its entire system with microwave communications going from tower to tower tens of kilometers apart. Even the company's name reflected this (MCI stood for Microwave Communications, Inc.). MCI has since gone over to fiber and merged with WorldCom.

Since the microwaves travel in a straight line, if the towers are too far apart, the earth will get in the way (think about a San Francisco to Amsterdam link). Consequently, repeaters are needed periodically. The higher the towers are, the farther apart they can be. The distance between repeaters goes up very roughly with the square root of the tower height. For 100-meter-high towers, repeaters can be spaced 80 km apart.

The Politics of the Electromagnetic Spectrum

The ISM bands in the United States



To prevent total chaos, there are national and international agreements about who gets to use which frequencies. Since everyone wants a higher data rate, everyone wants more spectrum. National governments allocate spectrum for AM and FM radio, television, and mobile phones, as well as for telephone companies, police, maritime, navigation, military, government, and many other competing users. Worldwide, an agency of ITU-R (WARC) tries to coordinate this allocation so devices that work in multiple countries can be manufactured. However, countries are not bound by ITU -R's recommendations, and the FCC (Federal Communication Commission), which does the allocation for the United States, has occasionally rejected ITU-R's recommendations (usually because they required some politically-powerful group giving up some piece of the spectrum).

Infrared and Millimeter Waves

Unguided infrared and millimeter waves are widely used for short-range communication. The remote controls used on televisions, VCRs, and stereos all use infrared communication. They are relatively directional, cheap, and easy to build but have a major drawback: they do not pass through solid objects (try standing between your remote control and your television and see if it still works). In general, as we go from long-wave radio toward visible light, the waves behave more and more like light and less and less like radio.

On the other hand, the fact that infrared waves do not pass through solid walls well is also a plus. It means that an infrared system in one room of a building will not interfere with a similar system in adjacent rooms or buildings: you cannot control your neighbor's television with your remote control. Furthermore, security of infrared systems against eavesdropping is better than that of radio

systems precisely for this reason. Therefore, no government license is needed to operate an infrared system, in contrast to radio systems, which must be licensed outside the ISM bands. Infrared communication has a limited use on the desktop, for example, connecting notebook computers and printers, but it is not a major player in the communication game.

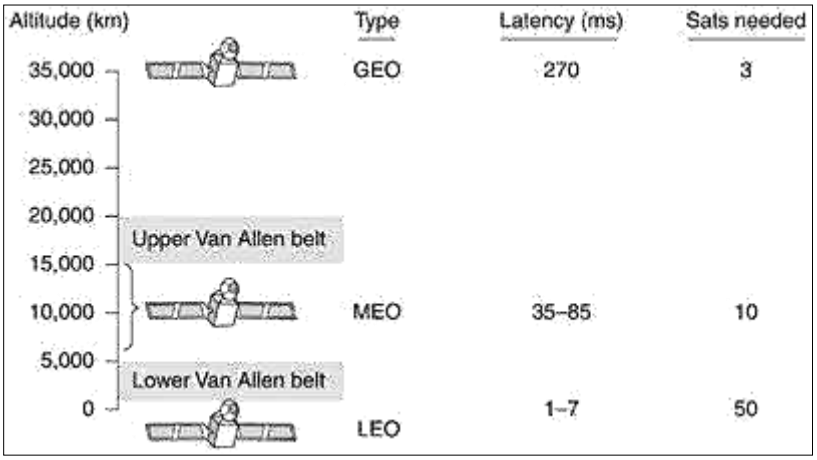
Light wave Transmission

Unguided optical signaling has been in use for centuries. Paul Revere used binary optical signaling from the Old North Church just prior to his famous ride. A more modern application is to connect the LANs in two buildings via lasers mounted on their rooftops. Coherent optical signaling using lasers is inherently unidirectional, so each building needs its own laser and its own photo detector. This scheme offers very high bandwidth and very low cost. It is also relatively easy to install and, unlike microwave, does not require an FCC license.

The laser's strength, a very narrow beam, is also its weakness here. Aiming a laser beam 1-mm wide at a target the size of a pin head 500 meters away requires the marksmanship of a latter-day Annie Oakley. Usually, lenses are put into the system to defocus the beam slightly.

A disadvantage is that laser beams cannot penetrate rain or thick fog, but they normally work well on sunny days. However, the author once attended a conference at a modern hotel in Europe at which the conference organizers thoughtfully provided a room full of terminals for the attendees to read their e-mail during boring presentations. Since the local PTT was unwilling to install a large number of telephone lines for just 3 days, the organizers put a laser on the roof and aimed it at their university's computer science building a few kilometers away. They tested it the night before the conference and it worked perfectly. At 9 a.m. the next morning, on a bright sunny day, the link failed completely and stayed down all day. That evening, the organizers tested it again very carefully, and once again it worked absolutely perfectly. The pattern repeated itself for two more days consistently.

Communication Satellites:



Communication satellites and some of their properties, including altitude above the earth, round-trip delay time, and number of satellites needed for global coverage

In the 1950s and early 1960s, people tried to set up communication systems by bouncing signals off metallized weather balloons. Unfortunately, the received signals were too weak to be of any practical use. Then the U.S. Navy noticed a kind of permanent weather balloon in the sky—the moon—and built an operational system for ship-to-shore communication by bouncing signals off it.

Further progress in the celestial communication field had to wait until the first communication satellite was launched. The key difference between an artificial satellite and a real one is that the artificial one can amplify the signals before sending them back, turning a strange curiosity into a powerful communication system.

Communication satellites have some interesting properties that make them attractive for many applications. In its simplest form, a communication satellite can be thought of as a big microwave repeater in the sky. It contains several transponders, each of which listens to some portion of the spectrum, amplifies the incoming signal, and then rebroadcasts it at another frequency to avoid interference with the incoming signal. The downward beams can be broad, covering a substantial fraction of the earth's surface, or narrow, covering an area only hundreds of kilometers in diameter. This mode of operation is known as a bent pipe.

According to Kepler's law, the orbital period of a satellite varies as the radius of the orbit to the 3/2 power. The higher the satellite, the longer the period.

Near the surface of the earth, the period is about 90 minutes. Consequently, low-orbit satellites pass out of view fairly quickly, so many of them are needed to provide continuous coverage. At an altitude of about 35,800 km, the period is 24 hours. At an altitude of 384,000 km, the period is about one month, as anyone who has observed the moon regularly can testify.

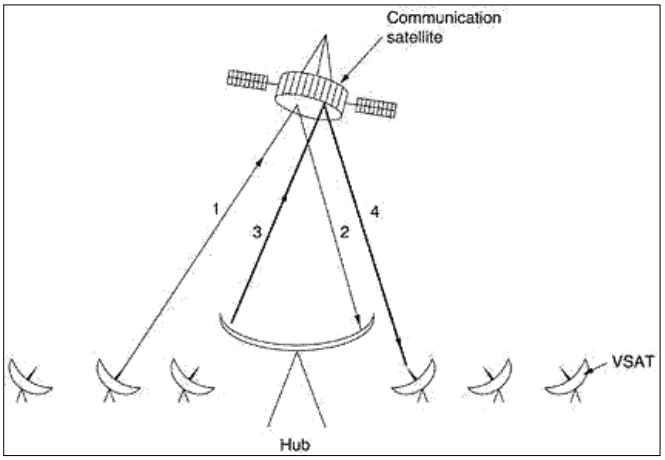
A satellite's period is important, but it is not the only issue in determining where to place it. Another issue is the presence of the Van Allen belts, layers of highly charged particles trapped by the earth's magnetic field. Any satellite flying within them would be destroyed fairly quickly by the highly-energetic charged particles trapped there by the earth's magnetic field. These factors lead to three regions in which satellites can be placed safely.

Geostationary Satellites

Band	Downlink	Uplink	Bandwidth	Problems
L	1.5 GHz	1.6 GHz	15 MHz	Low bandwidth; crowded
S	1.9 GHz	2.2 GHz	70 MHz	Low bandwidth; crowded
C	4.0 GHz	6.0 GHz	500 MHz	Terrestrial interference
Ku	11 GHz	14 GHz	500 MHz	Rain
Ka	20 GHz	30 GHz	3500 MHz	Rain, equipment cost

The principal satellite bands

In 1945, the science fiction writer Arthur C. Clarke calculated that a satellite at an altitude of 35,800 km in a circular equatorial orbit would appear to remain motionless in the sky. So it would not need to be tracked (Clarke, 1945). He went on to describe a complete communication system that used these (manned) geostationary satellites, including the orbits, solar panels, radio frequencies, and launch procedures. Unfortunately, he concluded that satellites were impractical due to the impossibility of putting power-hungry, fragile, vacuum tube amplifiers into orbit, so he never pursued this idea further, although he wrote some science fiction stories about it.



VSATs using a hub

The invention of the transistor changed all that, and the first artificial communication satellite, Telstar, was launched in July 1962. Since then, communication satellites have become a multibillion dollar business and the only aspect of outer space that has become highly profitable. These high-flying satellites are often called GEO (Geostationary Earth Orbit) satellites.

With current technology, it is unwise to have geostationary satellites spaced much closer than 2 degrees in the 360-degree equatorial plane, to avoid interference. With a spacing of 2 degrees, there can only be 360/2 = 180 of these satellites in the sky at once. However, each transponder can use multiple frequencies and polarizations to increase the available bandwidth.

To prevent total chaos in the sky, orbit slot allocation is done by ITU. This process is highly political, with countries barely out of the Stone Age demanding "their" orbit slots (for the purpose of leasing them to the highest bidder). Other countries, however, maintain that national property rights do not extend up to the moon and that no country has a legal right to the orbit slots above its territory. To add to the fight, commercial telecommunication is not the only application. Television broadcasters, governments, and the military also want a piece of the orbiting pie.

Modern satellites can be quite large, weighing up to 4000 kg and consuming several kilowatts of electric power produced by the solar panels. The effects of solar, lunar, and planetary gravity tend to move them away from their assigned orbit slots and orientations, an effect countered by on-board rocket motors. This fine-tuning activity is called station keeping. However, when the fuel for the motors has been exhausted, typically in about 10 years, the satellite drifts and tumbles helplessly, so it has to be turned off. Eventually, the orbit

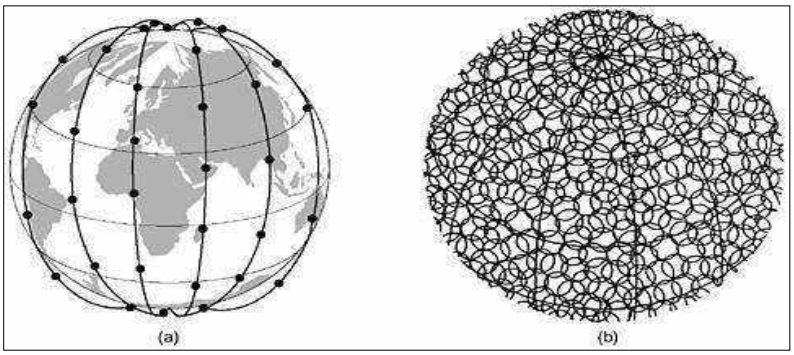
decays and the satellite reenters the atmosphere and burns up or occasionally crashes to earth.

Medium-Earth Orbit Satellites

At much lower altitudes, between the two Van Allen belts, we find the MEO (Medium-Earth Orbit) satellites. As viewed from the earth, these drift slowly in longitude, taking something like 6 hours to circle the earth. Accordingly, they must be tracked as they move through the sky. Because they are lower than the GEOs, they have a smaller footprint on the ground and require less powerful transmitters to reach them. Currently they are not used for telecommunications, so we will not examine them further here. The 24 GPS (Global Positioning System) satellites orbiting at about 18,000 km are examples of MEO satellites.

Low-Earth Orbit Satellites

Moving down in altitude, we come to the LEO (Low-Earth Orbit) satellites. Due to their rapid motion, large numbers of them are needed for a complete system. On the other hand, because the satellites are so close to the earth, the ground stations do not need much power, and the round-trip delay is only a few milliseconds. In this section we will examine three examples, two aimed at voice communication and one aimed at Internet service.



a) The Iridium satellites form six necklaces around the earth
(b) 1628 moving cells cover the earth

Iridium

As mentioned above, for the first 30 years of the satellite era, low-orbit satellites were rarely used because they zip into and out of view so quickly. In 1990, Motorola broke new ground by filing an application with the FCC asking for permission to launch 77 low-orbit satellites for the Iridium project (element 77 is iridium). The plan was later revised to use only 66 satellites, so the project

should have been renamed Dysprosium (element 66), but that probably sounded too much like a disease. The idea was that as soon as one satellite went out of view, another would replace it. This proposal set off a feeding frenzy among other communication companies. All of a sudden, everyone wanted to launch a chain of low-orbit satellites.

After seven years of cobbling together partners and financing, the partners launched the Iridium satellites in 1997. Communication service began in November 1998. Unfortunately, the commercial demand for large, heavy satellite telephones was negligible because the mobile phone network had grown spectacularly since 1990. As a consequence, Iridium was not profitable and was forced into bankruptcy in August 1999 in one of the most spectacular corporate fiascos in history. The satellites and other assets (worth $5 billion) were subsequently purchased by an investor for $25 million at a kind of extraterrestrial garage sale. The Iridium service was restarted in March 2001.

Iridium's business was (and is) providing worldwide telecommunication service using hand-held devices that communicate directly with the Iridium satellites. It provides voice, data, paging, fax, and navigation service everywhere on land, sea, and air. Customers include the maritime, aviation, and oil exploration industries, as well as people traveling in parts of the world lacking a telecommunications infrastructure (e.g., deserts, mountains, jungles, and some Third World countries).

The Iridium satellites are positioned at an altitude of 750 km, in circular polar orbits. They are arranged in north-south necklaces, with one satellite every 32 degrees of latitude. With six satellite necklaces, the entire earth is covered, as suggested by Fig. 2-18(a). People not knowing much about chemistry can think of this arrangement as a very, very big dysprosium atom, with the earth as the nucleus and the satellites as the electrons.

Global Star

An alternative design to Iridium is Global star. It is based on 48 LEO satellites but uses a different switching scheme than that of Iridium. Whereas Iridium relays calls from satellite to satellite, which requires sophisticated switching equipment in the satellites, Globalstar uses a traditional bent-pipe design. The call originating at the North Pole is sent back to earth and picked up by the large ground station at Santa's Workshop. The call is then routed via a terrestrial network to the ground station nearest the callee and delivered by a bent-pipe connection as shown. The advantage of this scheme is that it puts much of the complexity on the ground, where it is easier to manage. Also, the use of large

ground station antennas that can put out a powerful signal and receive a weak one means that lower-powered telephones can be used. After all, the telephone puts out only a few milliwatts of power, so the signal that gets back to the ground station is fairly weak, even after having been amplified by the satellite.

Teledesic

Iridium is targeted at telephone users located in odd places. Our next example, Teledesic, is targeted at bandwidth-hungry Internet users all over the world. It was conceived in 1990 by mobile phone pioneer Craig McCaw and Microsoft founder Bill Gates, who was unhappy with the snail's pace at which the world's telephone companies were providing high bandwidth to computer users. The goal of the Teledesic system is to provide millions of concurrent Internet users with an uplink of as much as 100 Mbps and a downlink of up to 720 Mbps using a small, fixed, VSAT-type antenna, completely bypassing the telephone system. To telephone companies, this is pie-in-the-sky.

The original design was for a system consisting of 288 small-footprint satellites arranged in 12 planes just below the lower Van Allen belt at an altitude of 1350 km. This was later changed to 30 satellites with larger footprints. Transmission occurs in the relatively uncrowded and high-bandwidth Ka band. The system is packet-switched in space, with each satellite capable of routing packets to its neighboring satellites. When a user needs bandwidth to send packets, it is requested and assigned dynamically in about 50 msec. The system is scheduled to go live in 2005 if all goes as planned.

Satellites versus Fiber

A comparison between satellite communication and terrestrial communication is instructive. As recently as 20 years ago, a case could be made that the future of communication lay with communication satellites. After all, the telephone system had changed little in the past 100 years and showed no signs of changing in the next 100 years. This glacial movement was caused in no small part by the regulatory environment in which the telephone companies were expected to provide good voice service at reasonable prices (which they did), and in return got a guaranteed profit on their investment. For people with data to transmit, 1200-bps modems were available. That was pretty much all there was.

The introduction of competition in 1984 in the United States and somewhat later in Europe changed all that radically. Telephone companies began replacing their long-haul networks with fiber and introduced high-bandwidth services like ADSL (Asymmetric Digital Subscriber Line). They also stopped their long-time

practice of charging artificially-high prices to long-distance users to subsidize local service.

All of a sudden, terrestrial fiber connections looked like the long-term winner. Nevertheless, communication satellites have some major niche markets that fiber does not (and, sometimes, cannot) address. We will now look at a few of these.

First, while a single fiber has, in principle, more potential bandwidth than all the satellites ever launched, this bandwidth is not available to most users. The fibers that are now being installed are used within the telephone system to handle many long distance calls at once, not to provide individual users with high bandwidth. With satellites, it is practical for a user to erect an antenna on the roof of the building and completely bypass the telephone system to get high bandwidth. Teledesic is based on this idea.

A second niche is for mobile communication. Many people nowadays want to communicate while jogging, driving, sailing, and flying. Terrestrial fiber optic links are of no use to them, but satellite links potentially are. It is possible, however, that a combination of cellular radio and fiber will do an adequate job for most users (but probably not for those airborne or at sea).

A third niche is for situations in which broadcasting is essential. A message sent by satellite can be received by thousands of ground stations at once. For example, an organization transmitting a stream of stock, bond, or commodity prices to thousands of dealers might find a satellite system to be much cheaper than simulating broadcasting on the ground.

A fourth niche is for communication in places with hostile terrain or a poorly developed terrestrial infrastructure. Indonesia, for example, has its own satellite for domestic telephone traffic. Launching one satellite was cheaper than stringing thousands of undersea cables among the 13,677 islands in the archipelago.

A fifth niche market for satellites is to cover areas where obtaining the right of way for laying fiber is difficult or unduly expensive.

Sixth, when rapid deployment is critical, as in military communication systems in time of war, satellites win easily.

In short, it looks like the mainstream communication of the future will be terrestrial fiber optics combined with cellular radio, but for some specialized uses, satellites are better. However, there is one caveat that applies to all of this: economics. Although fiber offers more bandwidth, it is certainly possible that terrestrial and satellite communication will compete aggressively on price. If advances in technology radically reduce the cost of deploying a satellite or low-

orbit satellites catch on in a big way, it is not certain that fiber will win in all markets.

UNIT - III
DATA LINK LAYER

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

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

Error-Correcting Codes

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

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

what the original block was, rather than relying on a retransmission, which itself may be in error.

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

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

```
10001001
10110001
00111000
```

The number of bit positions in which two code words differ is called the Hamming distance (Hamming, 1950). Its significance is that if two code words are a Hamming distance d apart, it will require d single-bit errors to convert one into the other.

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

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

As a simple example of an error -detecting code, consider a code in which a single parity bit is appended to the data. The parity bit is chosen so that the number of 1 bits in the code word is even (or odd). For example, when 1011010

is sent in even parity, a bit is added to the end to make it 10110100. With odd parity 1011010 becomes 10110101. A code with a single parity bit has a distance 2, since any single-bit error produces a code word with the wrong parity. It can be used to detect single errors.

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

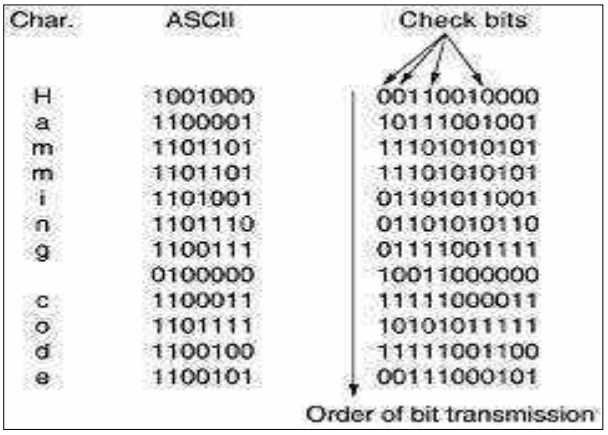
0000000000, 0000011111, 1111100000, and 1111111111

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

Use of a Hamming code to correct burst errors

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

This theoretical lower limit can, in fact, be achieved using a method due to Hamming (1950). The bits of the code word are numbered consecutively, starting with bit 1 at the left end, bit 2 to its immediate right, and so on. The bits that are powers of 2 (1, 2, 4, 8, 16, etc.) are check bits. The rest (3, 5, 6, 7, 9, etc.) are filled up with the m data bits. Each check bit forces the parity of some collection of bits, including itself, to be even (or odd). A bit may be included in several parity computations. To see which check bits the data bit in position k contributes to, rewrite k as a sum of powers of 2. For example, 11 = 1 + 2 + 8 and 29 = 1 + 4 + 8 + 16. A bit is checked by just those check bits occurring in its expansion (e.g., bit 11 is checked by bits 1, 2, and 8).



When a code word arrives, the receiver initializes a counter to zero. It then examines each check bit, k (k = 1, 2, 4, 8,), to see if it has the correct parity. If not, the receiver adds k to the counter. If the counter is zero after all the check bits have been examined (i.e., if they were all correct), the code word is accepted as valid. If the counter is nonzero, it contains the number of the incorrect bit. For example, if check bits 1, 2, and 8 are in error, the inverted bit is 11, because it is the only one checked by bits 1, 2, and 8. Shows some 7-bit ASCII characters encoded as 11-bit code words using a Hamming code. Remember that the data are found in bit positions 3, 5, 6, 7, 9, 10, and 11.

Hamming codes can only correct single errors. However, there is a trick that can be used to permit Hamming codes to correct burst errors. A sequence of k consecutive code words are arranged as a matrix, one code word per row. Normally, the data would be transmitted one code word at a time, from left to right. To correct burst errors, the data should be transmitted one column at a time, starting with the leftmost column. When all k bits have been sent, the second column is sent, When the frame arrives at the receiver, the matrix is reconstructed, one column at a time. If a burst error of length k occurs, at most 1 bit in each of the k code words will have been affected, but the Hamming code can correct one error per code word, so the entire block can be restored. This method uses kr check bits to make blocks of km data bits immune to a single burst error of length k or less.

Error-Detecting Codes

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

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

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

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

This method can detect a single burst of length n, since only 1 bit per column will be changed. A burst of length n1 will pass undetected, however, if the first bit is inverted, the last bit is inverted, and all the other bits are correct. (A burst error does not imply that all the bits are wrong; it just implies that at least the first and last are wrong.) If the block is badly garbled by a long burst or by multiple shorter bursts, the probability that any of the n columns will have the correct parity, by accident, is 0.5, so the probability of a bad block being accepted when it should not be is 2^{-n}.

Although the above scheme may sometimes be adequate, in practice, another method is in widespread use: the polynomial code, also known as a CRC (Cyclic Redundancy Check). Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from x^{k-1} to x^0. Such a polynomial is said to be of degree k - 1. The high-order (leftmost)

bit is the coefficient of x^{k-1}; the next bit is the coefficient of x^{k-2}, and so on. For example, 110001 has 6 bits and thus represents a six-term polynomial with coefficients 1, 1, 0, 0, 0, and 1: $x^5 + x^4 + x^0$.

Polynomial arithmetic is done modulo 2, according to the rules of algebraic field theory. There are no carries for addition or borrows for subtraction. Both addition and subtraction are identical to exclusive OR. For example:

Long division is carried out the same way as it is in binary except that the subtraction is done modulo 2, as above. A divisor is said "to go into" a dividend if the dividend has as many bits as the divisor.

When the polynomial code method is employed, the sender and receiver must agree upon a generator polynomial, G(x), in advance. Both the high-and low-order bits of the generator must be 1. To compute the checksum for some frame with m bits, corresponding to the polynomial M(x), the frame must be longer than the generator polynomial. The idea is to append a checksum to the end of the frame in such a way that the polynomial represented by the check summed frame is divisible by G(x). When the receiver gets the check summed frame, it tries dividing it by G(x). If there is a remainder, there has been a transmission error.

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

The algorithm for computing the checksum is as follows:

- Let r be the degree of G(x). Append r zero bits to the low-order end of the frame so it now contains m bits and corresponds to the polynomial $x^rM(x)$.
- Divide the bit string corresponding to G(x) into the bit string corresponding to $x^rM(x)$, using modulo 2 division.
- Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to $x^rM(x)$ using modulo 2 subtractions. The result is the check summed frame to be transmitted. Call its polynomial T(x).

Calculation of the polynomial code checksum

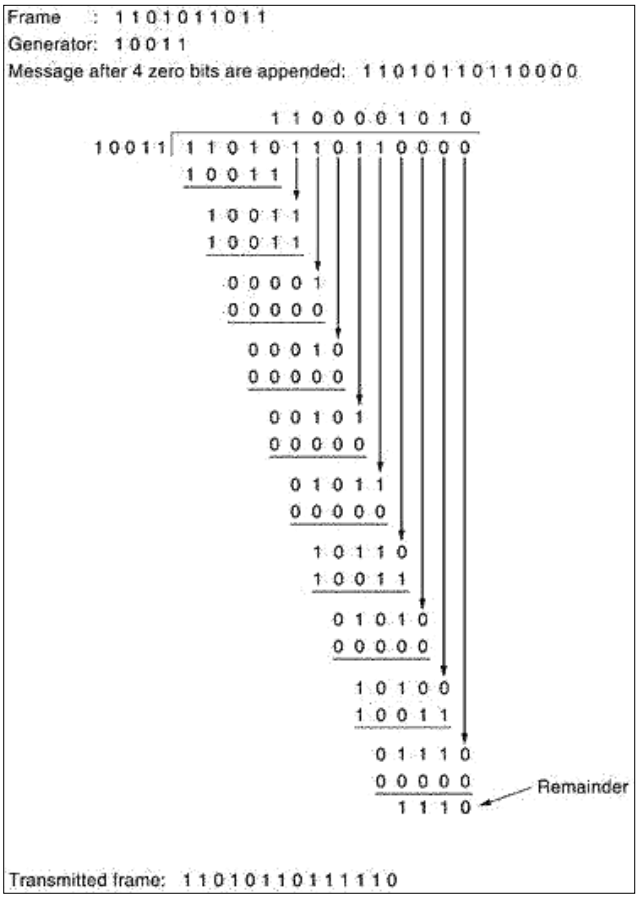


Figure 3-8illustrates the calculation for a frame 1101011011 using the generator $G(x) = x^4 + x + 1$.

It should be clear that T(x) is divisible (modulo 2) by G(x). In any division problem, if you diminish the dividend by the remainder, what is left over is divisible by the divisor. For example, in base 10, if you divide 210,278 by 10,941, the remainder is 2399. By subtracting 2399 from 210,278, what is left over (207,879) is divisible by 10,941.

Now let us analyze the power of this method. What kinds of errors will be detected? Imagine that a transmission error occurs, so that instead of the bit string for T(x) arriving, T(x) + E(x) arrives. Each 1 bit in E(x) corresponds to a bit that has been inverted. If there are k 1 bits in E(x), k single- bit errors have occurred.

A single burst error is characterized by an initial 1, a mixture of 0s and 1s, and a final 1, with all other bits being 0.

Upon receiving the check summed frame, the receiver divides it by G(x); that is, it computes [T(x) + E(x)]/G(x). T(x)/G(x) is 0, so the result of the computation is simply E(x)/G(x). Those errors that happen to correspond to polynomials containing G(x) as a factor will slip by; all other errors will be caught.

If there has been a single-bit error, $E(x) = x^i$, where i determines which bit is in error. If G(x) contains two or more terms, it will never divide E(x), so all single-bit errors will be detected.

If there have been two isolated single-bit errors, $E(x) = x^i + x^j$, where i > j. Alternatively, this can be written as $E(x) = x^j(x^{i - j} + 1)$. If we assume that G(x) is not divisible by x, a sufficient condition for all double errors to be detected is that G(x) does not divide $x^k + 1$ for any k up to the maximum value of i - j (i.e., up to the maximum frame length). Simple, low-degree polynomials that give protection to long frames are known. For example, $x^{15} + x^{14} + 1$ will not divide $x^k + 1$ for any value of k below 32,768.

If there are an odd number of bits in error, E(X) contains an odd number of terms (e.g., $x^5 + x^2 + 1$, but not $x^2 + 1$). Interestingly, no polynomial with an odd number of terms has x + 1 as a factor in the modulo 2 system. By making x + 1a factor of G(x), we can catch all errors consisting of an odd number of inverted bits.

To see that no polynomial with an odd number of terms is divisible by x + 1, assume that E(x) has an odd number of terms and is divisible by x + 1. Factor E(x) into (x + 1) Q(x). Now evaluate E(1) = (1 + 1)Q(1). Since 11 = 0 (modulo 2), E(1) must be zero. If E(x) has an odd number of terms, substituting 1 for x everywhere will always yield 1 as the result. Thus, no polynomial with an odd number of terms is divisible by x + 1.

Finally, and most importantly, a polynomial code with r check bits will detect all burst errors of length $\leq$r. A burst error of length k can be represented by $x^i(x^{k - 1} + ... + 1)$, where i determines how far from the right-hand end of the received frame the burst is located. If G(x) contains an x^0 term, it will not have x^i

as a factor, so if the degree of the parenthesized expression is less than the degree of G(x), the remainder can never be zero.

If the burst length is r + 1, the remainder of the division by G(x) will be zero if and only if the burst is identical to G(x). By definition of a burst, the first and last bits must be 1, so whether it matches depends on the r - 1 intermediate bits. If all combinations are regarded as equally likely, the probability of such an incorrect frame being accepted as valid is $½^{r-1}$.

It can also be shown that when an error burst longer than r + 1 bits occurs or when several shorter bursts occur, the probability of a bad frame getting through unnoticed is $½^{r}$, assuming that all bit patterns are equally likely.

Certain polynomials have become international standards. The one used in IEEE 802 is

Among other desirable properties, it has the property that it detects all bursts of length 32 or less and all bursts affecting an odd number of bits.

Although the calculation required to compute the checksum may seem complicated, Peterson and Brown (1961) have shown that a simple shift register circuit can be constructed to compute and verify the checksums in hardware. In practice, this hardware is nearly always used. Virtually all LANs use it and point-to-point lines do, too, in some cases.

For decades, it has been assumed that frames to be check summed contain random bits. All analyses of checksum algorithms have been made under this assumption. Inspection of real data has shown this assumption to be quite wrong. As a consequence, under some circumstances, undetected errors are much more common than had been previously thought (Partridge et al., 1995).

Elementary Data Link Protocols

To introduce the subject of protocols, we will begin by looking at three protocols of increasing complexity. For interested readers, a simulator for these and subsequent protocols is available via the Web (see the preface). Before we look at the protocols, it is useful to make explicit some of the assumptions underlying the model of communication. To start with, we assume that in the

physical layer, data link layer, and network layer are independent processes that communicate by passing messages back and forth. In many cases, the physical and data link layer processes will be running on a processor inside a special network I/O chip and the network layer code will be running on the main CPU. However, other implementations are also possible (e.g., three processes inside a single I/O chip; or the physical and data link layers as procedures called by the network layer process). In any event, treating the three layers as separate processes makes the discussion conceptually cleaner and also serves to emphasize the independence of the layers.

Some definitions needed in the protocols to follow. These definitions are located in the file protocol.h

```
#define MAX_PKT 1024                                /* determines packet size in bytes */

typedef enum {false, true} boolean;                 /* boolean type */
typedef unsigned int seq_nr;                        /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;/*    packet definition */
typedef enum {data, ack, nak} frame_kind;           /* frame_kind definition */

typedef struct {                                    /* frames are transported in this layer */
  frame_kind kind;                                  /* what kind of a frame is it? */
  seq_nr seq;                                       /* sequence number */
  seq_nr ack;                                       /* acknowledgement number */
  packet info;                                      /* the network layer packet */
} frame;

/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Another key assumption is that machine A wants to send a long stream of data to machine B, using a reliable, connection-oriented service. Later, we will consider the case where B also wants to send data to A simultaneously. A is assumed to have an infinite supply of data ready to send and never has to wait for data to be produced. Instead, when A's data link layer asks for data, the network

layer is always able to comply immediately. (This restriction, too, will be dropped later.)

We also assume that machines do not crash. That is, these protocols deal with communication errors, but not the problems caused by computers crashing and rebooting.

As far as the data link layer is concerned, the packet passed across the interface to it from the network layer is pure data, whose every bit is to be delivered to the destination's network layer. The fact that the destination's network layer may interpret part of the packet as a header is of no concern to the data link layer.

When the data link layer accepts a packet, it encapsulates the packet in a frame by adding a data link header and trailer to it Thus, a frame consists of an embedded packet, some control information (in the header), and a checksum (in the trailer). The frame is then transmitted to the data link layer on the other machine. We will assume that there exist suitable library procedures to_physical_layer to send a frame and from_physical_layer to receive a frame. The transmitting hardware computes and appends the checksum (thus creating the trailer), so that the datalink layer software need not worry about it. The polynomial algorithm discussed earlier in this chapter might be used, for example.

Initially, the receiver has nothing to do. It just sits around waiting for something to happen. In the example protocols of this chapter we will indicate that the data link layer is waiting for something to happen by the procedure call wait_for_event(&event). This procedure only returns when something has happened (e.g., a frame has arrived). Upon return, the variable event tells what happened. The set of possible events differs for the various protocols to be described and will be defined separately for each protocol. Note that in a more realistic situation, the data link layer will not sit in a tight loop waiting for an event, as we have suggested, but will receive an interrupt, which will cause it to stop whatever it was doing and go handle the incoming frame. Nevertheless, for simplicity we will ignore all the details of parallel activity within the data link layer and assume that it is dedicated full time to handling just our one channel.

When a frame arrives at the receiver, the hardware computes the checksum. If the checksum is incorrect (i.e., there was a transmission error), the data link layer is so informed (event = cksum_err). If the inbound frame arrived

undamaged, the data link layer is also informed (event = frame_ arrival) so that it can acquire the frame for inspection using from physical layer. As soon as the receiving data link layer has acquired an undamaged frame, it checks the control information in the header, and if everything is all right, passes the packet portion to the network layer. Under no circumstances is a frame header ever given to a network layer.

There is a good reason why the network layer must never be given any part of the frame header: to keep the network and data link protocols completely separate. As long as the network layer knows nothing at all about the data link protocol or the frame format, these things can be changed without requiring changes to the network layer's software. Providing a rigid interface between network layer and data link layer greatly simplifies the software design because communication protocols in different layers can evolve independently.

Figure 3-9shows some declarations (in C) common to many of the protocols to be discussed later. Five data structures are defined there: boolean, seq_nr, packet, frame_kind, and frame. A boolean is an enumerated type and can take on the values true and false. A seq_nr is a small integer used to number the frames so that we can tell them apart. These sequence numbers run from 0 up to and including MAX_SEQ, which is defined in each protocol needing it. A packet is the unit of information exchanged between the network layer and the data link layer on the same machine, or between network layer peers. In our model it always contains MAX_PKT bytes, but more realistically it would be of variable length.

These are library routines whose details are implementation dependent and whose inner workings will not concern us further here. The procedure wait_for_event sits in a tight loop waiting for something to happen, as mentioned earlier. The procedures to _network_layer and from_network_layer are used by the data link layer to pass packets to the network layer and accept packets from the network layer, respectively. Note that from_physical_ layer and to _physical_layer pass frames between the data link layer and physical layer. On the other hand, the procedures to_ network_layer and from_network_ layer pass packets between the data link layer and network layer. In other words, to_network_layer and from_network_layer deal with the interface between layers 2 and 3, whereas from_physical_layer and to_physical_layer deal with the interface between layers 1 and 2.

In most of the protocols, we assume that the channel is unreliable and loses entire frames upon occasion. To be able to recover from such calamities, the sending data link layer must start an internal timer or clock whenever it sends a frame. If no reply has been received within a certain predetermined time interval, the clock times out and the data link layer receives an interrupt signal.

In our protocols this is handled by allowing the procedure wait_for_event to return event = timeout. The procedures start_timer and stop_timer turn the timer on and off, respectively. Timeouts are possible only when the timer is running. It is explicitly permitted to call start_timer while the timer is running; such a call simply resets the clock to cause the next timeout after a full timer interval has elapsed (unless it is reset or turned off in the meanwhile).

The procedures start_ack_timer and stop_ack_timer control an auxiliary timer used to generate acknowledgements under certain conditions.

The procedures enable_network_layer and disable_network_layer are used in the more sophisticated protocols, where we no longer assume that the network layer always has packets to send. When the data link layer enables the network layer, the network layer is then permitted to interrupt when it has a packet to be sent. We indicate this with event = network_layer_ready. When a network layer is disabled, it may not cause such events. By being careful about when it enables and disables its network layer, the data link layer can prevent the network layer from swamping it with packets for which it has no buffer space.

Frame sequence numbers are always in the range 0 to MAX_SEQ (inclusive), where MAX_SEQ is different for the different protocols. It is frequently necessary to advance a sequence number by 1 circularly (i.e., MAX_SEQ is followed by 0). The macro inc performs this incrementing. It has been defined as a macro because it is used in-line within the critical path. As we will see later, the factor limiting network performance is often protocol processing, so defining simple operations like this as macros does not affect the readability of the code but does improve performance. Also, since MAX_ SEQ will have different values in different protocols, by making it a macro, it becomes possible to include all the protocols in the same binary without conflict. This ability is useful for the simulator.

The declarations of are part of each of the protocols to follow. To save space and to provide a convenient reference, they have been extracted and listed

together, but conceptually they should be merged with the protocols themselves. In C, this merging is done by putting the definitions in a special header file, in this case protocol's, and using the #include facility of the C preprocessor to include them in the protocol files.

A frame is composed of four fields: kind, seq, ack, and info, the first three of which contain control information and the last of which may contain actual data to be transferred. These control fields are collectively called the frame header.

The kind field tells whether there are any data in the frame, because some of the protocols distinguish frames containing only control information from those containing data as well. The seq and ack fields are used for sequence numbers and acknowledgements, respectively; their use will be described in more detail later. The info field of a data frame contains a single packet; the info field of a control frame is not used. A more realistic implementation would use a variable-length info field, omitting it altogether for control frames.

Again, it is important to realize the relationship between a packet and a frame. The network layer builds a packet by taking a message from the transport layer and adding the network layer header to it. This packet is passed to the data link layer for inclusion in the info field of an outgoing frame. When the frame arrives at the destination, the data link layer extracts the packet from the frame and passes the packet to the network layer. In this manner, the network layer can act as though machines can exchange packets directly.

An Unrestricted Simplex Protocol

As an initial example we will consider a protocol that is as simple as it can be. Data are transmitted in one direction only. Both the transmitting and receiving network layers are always ready. Processing time can be ignored. Infinite buffer space is available. And best of all, the communication channel between the data link layers never damages or loses frames. This thoroughly unrealistic protocol, which we will nickname "utopia,"

The protocol consists of two distinct procedures, a sender and a receiver. The sender runs in the data link layer of the source machine, and the receiver runs in the data link layer of the destination machine. No sequence numbers or

acknowledgements are used here, so MAX_SEQ is not needed. The only event type possible is frame_arrival (i.e., the arrival of an undamaged frame).

The sender is in an infinite while loop just pumping data out onto the line as fast as it can. The body of the loop consists of three actions: go fetch a packet from the (always obliging) network layer, construct an outbound frame using the variable s, and send the frame on its way. Only the info field of the frame is used by this protocol, because the other fields have to do with error and flow control and there are no errors or flow control restrictions here.

An unrestricted simplex protocol

```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
   sender to receiver.  The communication channel is assumed to be error free
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                        /* buffer for an outbound frame */
  packet buffer;                  /* buffer for an outbound packet */

  while (true) {
      from_network_layer(&buffer);  /* go get something to send */
      s.info = buffer;              /* copy it into s for transmission */
      to_physical_layer(&s);        /* send it on its way */
  }                                 /* Tomorrow, and tomorrow, and tomorrow,
                                       Creeps in this petty pace from day to day
                                       To the last syllable of recorded time.
                                           - Macbeth, V, v */
}


void receiver1(void)
{
  frame r;
  event_type event;               /* filled in by wait, but not used here */

  while (true) {
      wait_for_event(&event);       /* only possibility is frame_arrival */
      from_physical_layer(&r);      /* go get the inbound frame */
      to_network_layer(&r.info);    /* pass the data to the network layer */
  }
}
```

The receiver is equally simple. Initially, it waits for something to happen, the only possibility being the arrival of an undamaged frame. Eventually, the frame arrives and the procedure wait_for_event returns, with event set to frame_arrival (which is ignored anyway). The call to from_ physical_layer removes the newly arrived frame from the hardware buffer and puts it in the variable r, where the receiver code can get at it. Finally, the data portion is passed on to the network layer, and the data link layer settles back to wait for the next frame, effectively suspending itself until the frame arrives.

A Simplex Stop-and-Wait Protocol

Now we will drop the most unrealistic restriction used in protocol 1: the ability of the receiving network layer to process incoming data infinitely quickly (or equivalently, the presence in the receiving data link layer of an infinite amount of buffer space in which to store all incoming frames while they are waiting their respective turns). The communication channel is still assumed to be error free however, and the data traffic is still simplex.

The main problem we have to deal with here is how to prevent the sender from flooding the receiver with data faster than the latter is able to process them. In essence, if the receiver requires a time t to execute from_physical_layer plus to_network_layer, the sender must transmit at an average rate less than one frame per time t. Moreover, f we assume that no automatic buffering and queueing are done within the receiver's hardware, until the old one has been fetched by from_physical_layer, lest the new one overwrite the old one.

A simplex stop-and-wait protocol

```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                          /* buffer for an outbound frame */
  packet buffer;                    /* buffer for an outbound packet */
  event_type event;                 /* frame_arrival is the only possibility */

  while (true) {
      from_network_layer(&buffer);  /* go get something to send */
      s.info = buffer;              /* copy it into s for transmission */
      to_physical_layer(&s);        /* bye-bye little frame */
      wait_for_event(&event);       /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                       /* buffers for frames */
  event_type event;                 /* frame_arrival is the only possibility */
  while (true) {
      wait_for_event(&event);       /* only possibility is frame_arrival */
      from_physical_layer(&r);      /* go get the inbound frame */
      to_network_layer(&r.info);    /* pass the data to the network layer */
      to_physical_layer(&s);        /* send a dummy frame to awaken sender */
  }
}
```

In certain restricted circumstances (e.g., synchronous transmission and a receiving data link layer fully dedicated to processing the one input line), it might be possible for the sender to simply insert a delay into protocol 1 to slow it down sufficiently to keep from swamping the receiver. However, more usually, each data link layer will have several lines to attend to, and the time interval between

a frame arriving and its being processed may vary considerably. If the network designers can calculate the worst-case behavior of the receiver, they can program the sender to transmit so slowly that even if every frame suffers the maximum delay, there will be no overruns. The trouble with this approach is that it is too conservative. It leads to a bandwidth utilization that is far below the optimum, unless the best and worst cases are almost the same (i.e., the variation in the data link layer's reaction time is small).

A more general solution to this dilemma is to have the receiver provide feedback to the sender. After having passed a packet to its network layer, the receiver sends a little dummy frame back to the sender which, in effect, gives the sender permission to transmit the next frame. After having sent a frame, the sender is required by the protocol to bide its time until the little dummy (i.e., acknowledgement) frame arrives. Using feedback from the receiver to let the sender know when it may send more data is an example of the flow control mentioned earlier.

Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called stop-and-wait.

Although data traffic in this example is simplex, going only from the sender to the receiver, frames do travel in both directions. Consequently, the communication channel between the two data link layers needs to be capable of bidirectional information transfer. However, this protocol entails a strict alternation of flow: first the sender sends a frame, then the receiver sends a frame, then the sender sends another frame, then the receiver sends another one, and so on. A half- duplex physical channel would suffice here.

As in protocol 1, the sender starts out by fetching a packet from the network layer, using it to construct a frame, and sending it on its way. But now, unlike in protocol 1, the sender must wait until an acknowledgement frame arrives before looping back and fetching the next packet from the network layer. The sending data link layer need not even inspect the incoming frame: there is only one possibility. The incoming frame is always an acknowledgement.

The only difference between receiver1 and receiver2 is that after delivering a packet to the network layer, receiver2 sends an acknowledgement frame back to the sender before entering the wait loop again. Because only the arrival of the

frame back at the sender is important, not its contents, the receiver need not put any particular information in it.

A Simplex Protocol for a Noisy Channel

Now let us consider the normal situation of a communication channel that makes errors. Frames may be either damaged or lost completely. However, we assume that if a frame is damaged in transit, the receiver hardware will detect this when it computes the checksum. If the frame is damaged in such a way that the checksum is nevertheless correct, an unlikely occurrence, this protocol (and all other protocols) can fail (i.e., deliver an incorrect packet to the network layer).

At first glance it might seem that a variation of protocol 2 would work: adding a timer. The sender could send a frame, but the receiver would only send an acknowledgement frame if the data were correctly received. If a damaged frame arrived at the receiver, it would be discarded. After a while the sender would time out and send the frame again. This process would be repeated until the frame finally arrived intact.

The above scheme has a fatal flaw in it. Think about the problem and try to discover what might go wrong before reading further.

To see what might go wrong, remember that it is the task of the data link layer processes to provide error-free, transparent communication between network layer processes. The network layer on machine A gives a series of packets to its data link layer, which must ensure that an identical series of packets are delivered to the network layer on machine B by its data link layer. In particular, the network layer on B has no way of knowing that a packet has been lost or duplicated, so the data link layer must guarantee that no combination of transmission errors, however unlikely, can cause a duplicate packet to be delivered to a network layer.

A positive acknowledgement with retransmission protocol.

Consider the following scenario:

- The network layer on A gives packet 1 to its data link layer. The packet is correctly received at B and passed to the network layer on B. B sends an acknowledgement frame back to A.

- The acknowledgement frame gets lost completely. It just never arrives at all. Life would be a great deal simpler if the channel mangled and lost only data frames and not control frames, but sad to say, the channel is not very discriminating.
- The data link layer on A eventually times out. Not having received an acknowledgement, it (incorrectly) assumes that its data frame was lost or damaged and sends the frame containing packet 1 again.
- The duplicate frame also arrives at the data link layer on B perfectly and is unwittingly passed to the network layer there. If A is sending a file to B, part of the file will be duplicated (i.e., the copy of the file made by B will be incorrect and the error will not have been detected). In other words, the protocol will fail.

Clearly, what is needed is some way for the receiver to be able to distinguish a frame that it is seeing for the first time from a retransmission. The obvious way to achieve this is to have the sender put a sequence number in the header of each frame it sends. Then the receiver can check the sequence number of each arriving frame to see if it is a new frame or a duplicate to be discarded.

Since a small frame header is desirable, the question arises: What is the minimum number of bits needed for the sequence number? The only ambiguity in this protocol is between a frame, m, and its direct successor, m + 1. If frame m is lost or damaged, the receiver will not acknowledge it, so the sender will keep trying to send it. Once it has been correctly received, the receiver will send an acknowledgement to the sender. It is here that the potential trouble crops up. Depending upon whether the acknowledgement frame gets back to the sender correctly or not, the sender may try to send m or m + 1.

The event that triggers the sender to start sending frame m + 2 is the arrival of an acknowledgement for frame m

But this implies that m has been correctly received, and furthermore that its acknowledgement has also been correctly received by the sender (otherwise, the sender would not have begun with m + 1, let alone m + 2). As a consequence, the only ambiguity is between a frame and its immediate predecessor or successor, not between the predecessor and successor themselves.

A 1-bit sequence number (0 or 1) is therefore sufficient. At each instant of time, the receiver expects a particular sequence number next. Any arriving frame

containing the wrong sequence number is rejected as a duplicate. When a frame containing the correct sequence number arrives, it is accepted and passed to the network layer. Then the expected sequence number is incremented modulo 2 (i.e., 0 becomes 1 and 1 becomes 0).

Protocols in which the sender waits for a positive acknowledgement before advancing to the next data item are often called PAR (Positive Acknowledgement with Retransmission) or ARQ (Automatic Repeat reQuest). Like protocol 2, this one also transmits data only in one direction.

Protocol 3 differs from its predecessors in that both sender and receiver have a variable whose value is remembered while the data link layer is in the wait state. The sender remembers the sequence number of the next frame to send in next_frame_to_send; the receiver remembers the sequence number of the next frame expected in frame_expected. Each protocol has a short initialization phase before entering the infinite loop.

After transmitting a frame, the sender starts the timer running. If it was already running, it will be reset to allow another full timer interval. The time interval should be chosen to allow enough time for the frame to get to the receiver, for the receiver to process it in the worst case, and for the acknowledgement frame to propagate back to the sender. Only when that time interval has elapsed is it safe to assume that either the transmitted frame or its acknowledgement has been lost, and to send a duplicate. If the timeout interval is set too short, the sender will transmit unnecessary frames. While these extra frames will not affect the correctness of the protocol, they will hurt performance.

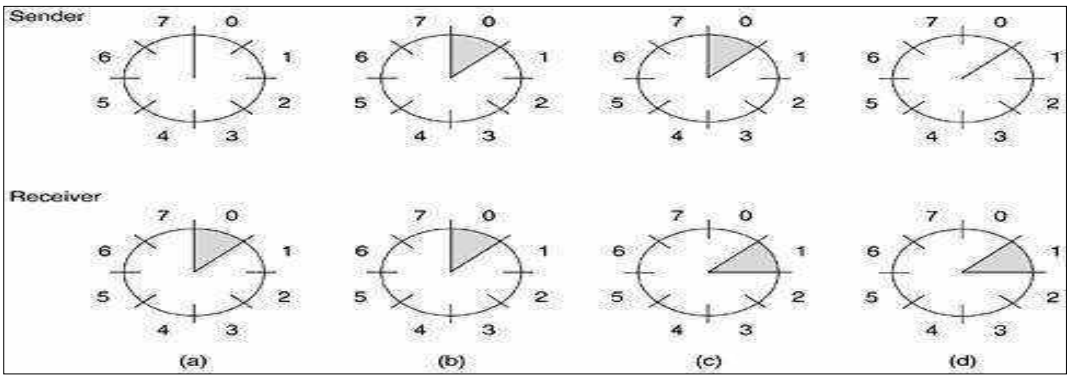
After transmitting a frame and starting the timer, the sender waits for something exciting to happen. Only three possibilities exist: an acknowledgement frame arrives undamaged, a damaged acknowledgement frame staggers in, or the timer expires. If a valid acknowledgement comes in, the sender fetches the next packet from its network layer and puts it in the buffer, overwriting the previous packet. It also advances the sequence number. If a damaged frame arrives or no frame at all arrives, neither the buffer nor the sequence number is changed so that a duplicate can be sent.

When a valid frame arrives at the receiver, its sequence number is checked to see if it is a duplicate. If not, it is accepted, passed to the network layer, and an

acknowledgement is generated. Duplicates and damaged frames are not passed to the network layer.

Sliding Window Protocols

In the previous protocols, data frames were transmitted in one direction only. In most practical situations, there is a need for transmitting data in both directions. One way of achieving full-duplex data transmission is to have two separate communication channels and use each one for simplex data traffic (in different directions). If this is done, we have two separate physical circuits, each with a "forward" channel (for data) and a "reverse" channel (for acknowledgements). In both cases the bandwidth of the reverse channel is almost entirely wasted. In effect, the user is paying for two circuits but using only the capacity of one.



A sliding window of size 1, with a 3-bit sequence number
(a) Initially (b) After the first frame has been sent (c) After the first frame has been received. (d) After the first acknowledgement has been received

A better idea is to use the same circuit for data in both directions. After all, in protocols 2 and 3 it was already being used to transmit frames both ways, and the reverse channel has the same capacity as the forward channel. In this model the data frames from A to B are intermixed with the acknowledgement frames from A to B. By looking at the kind field in the header of an incoming frame, the receiver can tell whether the frame is data or acknowledgement.

Although interleaving data and control frames on the same circuit is an improvement over having two separate physical circuits, yet another improvement is possible. When a data frame arrives, instead of immediately

sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet. The acknowledgement is attached to the outgoing data frame (using the ack field in the frame header). In effect, the acknowledgement gets a free ride on the next outgoing data frame. The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as piggybacking.

The principal advantage of using piggybacking over having distinct acknowledgement frames is a better use of the available channel bandwidth. The ack field in the frame header costs only a few bits, whereas a separate frame would need a header, the acknowledgement, and a checksum. In addition, fewer frames sent means fewer "frame arrival" interrupts, and perhaps fewer buffers in the receiver, depending on how the receiver's software is organized. In the next protocol to be examined, the piggyback field costs only 1 bit in the frame header. It rarely costs more than a few bits.

However, piggybacking introduces a complication not present with separate acknowledgements. How long should the data link layer wait for a packet onto which to piggyback the acknowledgement? If the data link layer waits longer than the sender's timeout period, the frame will be retransmitted, defeating the whole purpose of having acknowledgements. If the data link layer were an oracle and could foretell the future, it would know when the next network layer packet was going to come in and could decide either to wait for it or send a separate acknowledgement immediately, depending on how long the projected wait was going to be. Of course, the data link layer cannot foretell the future, so it must resort to some ad hoc scheme, such as waiting a fixed number of milliseconds. If a new packet arrives quickly, the acknowledgement is piggybacked onto it; otherwise, if no new packet has arrived by the end of this time period, the data link layer just sends a separate acknowledgement frame.

The next three protocols are bidirectional protocols that belong to a class called sliding window protocols. The three differ among themselves in terms of efficiency, complexity, and buffer requirements, as discussed later. In these, as in all sliding window protocols, each outbound frame contains a sequence number, ranging from 0 up to some maximum. The maximum is usually $2^n - 1$ so the sequence number fits exactly in an n-bit field. The stop-and-wait sliding window protocol uses n = 1, restricting the sequence numbers to 0 and 1, but more sophisticated versions can use arbitrary n.

Since frames currently within the sender's window may ultimately be lost or damaged in transit, the sender must keep all these frames in its memory for possible retransmission. Thus, if the maximum window size is n, the sender needs n buffers to hold the unacknowledged frames. If the window ever grows to its maximum size, the sending data link layer must forcibly shut off the network layer until another buffer becomes free.

The receiving data link layer's window corresponds to the frames it may accept. Any frame falling outside the window is discarded without comment. When a frame whose sequence number is equal to the lower edge of the window is received, it is passed to the network layer, an acknowledgement is generated, and the window is rotated by one. Unlike the sender's window, the receiver's window always remains at its initial size. Note that a window size of 1 means that the data link layer only accepts frames in order, but for larger windows this is not so. The network layer, in contrast, is always fed data in the proper order, regardless of the data link layer's window size.

Shows an example with a maximum window size of 1. Initially, no frames are outstanding, so thelower and upper edges of the sender's window are equal, but as time goes on, the situation progresses as shown.

A One-Bit Sliding Window Protocol

Before tackling the general case, let us first examine a sliding window protocol with a maximum window size ofSuch a protocol uses stop-and-wait since the sender transmits a frame and waits for its acknowledgement before sending the next one.

```
/* Protocol 4 (sliding window) is bidirectional. */

#define MAX_SEQ 1                       /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;            /* 0 or 1 only */
  seq_nr frame_expected;                /* 0 or 1 only */
  frame r, s;                           /* scratch variables */
  packet buffer;                        /* current packet being sent */
  event_type event;

  next_frame_to_send = 0;               /* next frame on the outbound stream */
  frame_expected = 0;                   /* frame expected next */
  from_network_layer(&buffer);          /* fetch a packet from the network layer */
  s.info = buffer;                      /* prepare to send the initial frame */
  s.seq = next_frame_to_send;           /* insert sequence number into frame */
  s.ack = 1 – frame_expected;           /* piggybacked ack */
  to_physical_layer(&s);                /* transmit the frame */
  start_timer(s.seq);                   /* start the timer running */
  while (true) {
      wait_for_event(&event);           /* frame_arrival, cksum_err, or timeout */
      if (event == frame_arrival) {     /* a frame has arrived undamaged. */
          from_physical_layer(&r);      /* go get it */
          if (r.seq == frame_expected) {/* handle inbound frame stream. */
              to_network_layer(&r.info);/* pass packet to network layer */
              inc(frame_expected);      /* invert seq number expected next */
          }
          if (r.ack == next_frame_to_send) {/* handle outbound frame stream. */
              stop_timer(r.ack);        /* turn the timer off */
              from_network_layer(&buffer);/* fetch new pkt from network layer */
              inc(next_frame_to_send);  /* invert sender's sequence number */
          }
      }
      s.info = buffer;                  /* construct outbound frame */
      s.seq = next_frame_to_send;       /* insert sequence number into it */
      s.ack = 1 – frame_expected;       /* seq number of last received frame */
      to_physical_layer(&s);            /* transmit a frame */
      start_timer(s.seq);               /* start the timer running */
  }
}
```

A 1-bit sliding window protocol

It starts out by defining some variables.Next_frame_to_send tells which frame the sender is trying to send. Similarly, frame_expected tells which frame the receiver is expecting. In both cases, 0 and 1 are the only possibilities.

Under normal circumstances, one of the two data link layers goes first and transmits the first frame. In other words, only one of the data link layer programs should contain the to_physical_layer and start_timer procedure calls outside the main loop. In the event that both data link layers start off simultaneously, a peculiar situation arises, as discussed later. The starting machine fetches the first packet from its network layer, builds a frame from it, and sends it. When this (or any) frame arrives, the receiving data link layer checks to see if it is a duplicate, just as in protocol 3. If the frame is the one expected, it is passed to the network layer and the receiver's window is slid up.

The acknowledgement field contains the number of the last frame received without error. If this number agrees with the sequence number of the frame the sender is trying to send, the sender knows it is done with the frame stored in bufferand can fetch the next packet from its network layer.

Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet

If the sequence number disagrees, it must continue trying to send the same frame. Whenever a frame is received, a frame is also sent back.

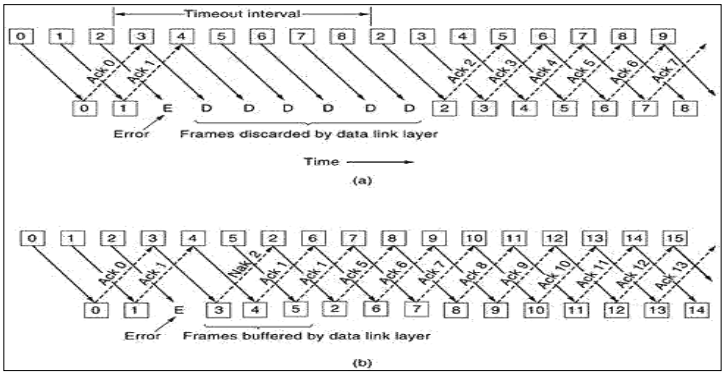
Now let us examine protocol 4 to see how resilient it is to pathological scenarios. Assume that computer A is trying to send its frame 0 to computer B and that B is trying to send its frame 0 to A. Suppose that A sends a frame to B, but A's timeout interval is a little too short. Consequently, A may time out repeatedly, sending a series of identical frames, all with seq = 0 and ack = 1.

When the first valid frame arrives at computer B, it will be accepted and frame_expected will be set to 1. All the subsequent frames will be rejected because B is now expecting frames with sequence number 1, not 0. Furthermore, since all the duplicates have ack = 1 and B is still waiting for an acknowledgement of 0, B will not fetch a new packet from its network layer.

After every rejected duplicate comes in, B sends A a frame containing seq = 0 and ack = 0. Eventually, one of these arrives correctly at A, causing A to begin sending the next packet. No combination of lost frames or premature timeouts can cause the protocol to deliver duplicate packets to either network layer, to skip a packet, or to deadlock.

However, a peculiar situation arises if both sides simultaneously send an initial packet. In part (a), the normal operation of the protocol is shown. In (b) the peculiarity is illustrated. If B waits for A's first frame before sending one of its own, the sequence is as shown in (a), and every frame is accepted. However, if A and B simultaneously initiate communication, their first frames cross, and the data link layers then get into situation (b). In (a) each frame arrival brings a new packet for the network layer; there are no duplicates. In (b) half of the frames contain duplicates, even though there are no transmission errors. Similar situations can occur as a result of premature timeouts, even when one side clearly

starts first. In fact, if multiple premature timeouts occur, frames may be sent three or more times.



Pipelining and error recovery

Effect of an error when (a) receiver's window size is 1 and (b) receiver's window size is large

A Protocol Using Go Back N

Until now we have made the tacit assumption that the transmission time required for a frame to arrive at the receiver plus the transmission time for the acknowledgement to come back is negligible. Sometimes this assumption is clearly false. In these situations the long round-trip time can have important implications for the efficiency of the bandwidth utilization. As an example, consider a 50-kbps satellite channel with a 500-msec round-trip propagation delay. Let us imagine trying to use protocol 4 to send 1000-bit frames via the satellite. At t = 0 the sender starts sending the first frame. At t = 20 msec the frame has been completely sent. Not until t = 270 msec has the frame fully arrived at the receiver, and not until t = 520 msec has the acknowledgement arrived back at the sender, under the best of circumstances (no waiting in the receiver and a short acknowledgement frame). This means that the sender was blocked during 500/520 or 96 percent of the time. In other words, only 4 percent of the available bandwidth was used. Clearly, the combination of a long transit time, high bandwidth, and short frame length is disastrous in terms of efficiency.

The problem described above can be viewed as a consequence of the rule requiring a sender to wait for an acknowledgement before sending another frame. If we relax that restriction, much better efficiency can be achieved. Basically, the solution lies in allowing the sender to transmit up to w frames before blocking, instead of just 1. With an appropriate choice of w the sender will be able to continuously transmit frames for a time equal to the round-trip transit time without filling up the window. In the example above, w should be at least 26. The

sender begins sending frame 0 as before. By the time it has finished sending 26 frames, at t = 520, the acknowledgement for frame 0 will have just arrived. Thereafter, acknowledgements arrive every 20 msec, so the sender always gets permission to continue just when it needs it. At all times, 25 or 26 unacknowledged frames are outstanding. Put in other terms, the sender's maximum window size is 26.

The need for a large window on the sending side occurs whenever the product of bandwidth x round-trip-delay is large. If the bandwidth is high, even for a moderate delay, the sender will exhaust its window quickly unless it has a large window. If the delay is high (e.g., on a geostationary satellite channel), the sender will exhaust its window even for a moderate bandwidth. The product of these two factors basically tells what the capacity of the pipe is, and the sender needs the ability to fill it without stopping in order to operate at peak efficiency.

This technique is known as pipelining. If the channel capacity is b bits/sec, the frame size l bits, and the round-trip propagation time R sec, the time required to transmit a single frame is l/b sec. After the last bit of a data frame has been sent, there is a delay of R/2 before that bit arrives at the receiver and another delay of at least R/2 for the acknowledgement to come back, for a total delay of R. In stop-and-wait the line is busy for l/band idle for R, giving

$$\text{line utilization} = l/(l + bR)$$

If l <bR, the efficiency will be less than 50 percent. Since there is always a nonzero delay for the acknowledgement to propagate back, pipelining can, in principle, be used to keep the line busy during this interval, but if the interval is small, the additional complexity is not worth the trouble.

Pipelining frames over an unreliable communication channel raises some serious issues. First, what happens if a frame in the middle of a long stream is damaged or lost? Large numbers of succeeding frames will arrive at the receiver before the sender even finds out that anything is wrong. When a damaged frame arrives at the receiver, it obviously should be discarded, but what should the receiver do with all the correct frames following it? Remember that the receiving data link layer is obligated to hand packets to the network layer in sequence.

Two basic approaches are available for dealing with errors in the presence of pipelining. One way, called go back n, is for the receiver simply to discard all

subsequent frames, sending no acknowledgements for the discarded frames. This strategy corresponds to a receive window of size 1. In other words, the data link layer refuses to accept any frame except the next one it must give to the network layer. If the sender's window fills up before the timer runs out, the pipeline will begin to empty. Eventually, the sender will time out and retransmit all unacknowledged frames in order, starting with the damaged or lost one. This approach can waste a lot of bandwidth if the error rate is high.

In Fig. 3-16(a) we see go back n for the case in which the receiver's window is large. Frames 0 and 1 are correctly received and acknowledged. Frame 2, however, is damaged or lost. The sender, unaware of this problem, continues to send frames until the timer for frame 2 expires. Then it backs up to frame 2 and starts all over with it, sending 2, 3, 4, etc. all over again.

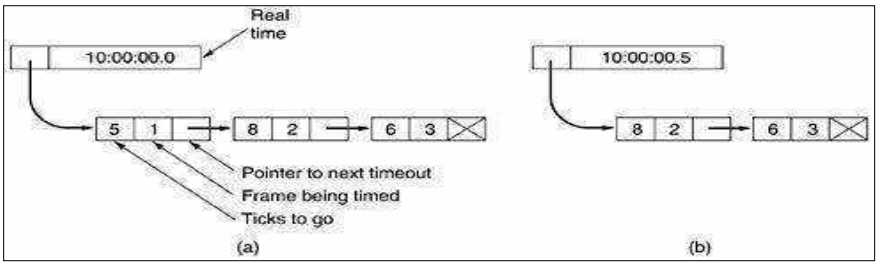
The other general strategy for handling errors when frames are pipelined is called selective repeat. When it is used, a bad frame that is received is discarded, but good frames received after it are buffered. When the sender times out, only the oldest unacknowledged frame is retransmitted. If that frame arrives correctly, the receiver can deliver to the network layer, in sequence, all the frames it has buffered. Selective repeat is often combined with having the receiver send a negative acknowledgement (NAK) when it detects an error, for example, when it receives a checksum error or a frame out of sequence. NAKs stimulate retransmission before the corresponding timer expires and thus improve performance.

In frames 0 and 1 are again correctly received and acknowledged and frame 2 is lost. When frame 3 arrives at the receiver, the data link layer there notices that is has missed a frame, so it sends back a NAK for 2 but buffers 3. When frames 4 and 5 arrive, they, too, are buffered by the data link layer instead of being passed to the network layer. Eventually, the NAK 2 gets back to the sender, which immediately resends frame 2. When that arrives, the data link layer now has 2, 3, 4, and 5 and can pass all of them to the network layer in the correct order. It can also acknowledge all frames up to and including 5, as shown in the figure. If the NAK should get lost, eventually the sender will time out for frame 2 and send it (and only it) of its own accord, but that may be a quite a while later. In effect, the NAK speeds up the retransmission of one specific frame.

Selective repeat corresponds to a receiver window larger than 1. Any frame within the window may be accepted and buffered until all the preceding ones have been passed to the network layer. This approach can require large amounts of data link layer memory if the window is large.

These two alternative approaches are trade-offs between bandwidth and data link layer buffer space. Depending on which resource is scarcer, one or the other can be used. Shows a pipelining protocol in which the receiving data link layer only accepts frames in order; frames following an error are discarded. In this protocol, for the first time we have dropped the assumption that the network layer always has an infinite supply of packets to send. When the network layer has a packet it wants to send, it can cause a network_layer_ready event to happen. However, to enforce the flow control rule of no more than MAX_SEQ unacknowledged frames outstanding at any time, the data link layer must be able to keep the network layer from bothering it with more work. The library procedures enable_network_layer and disable_network_layer do this job.

Note that a maximum of MAX_SEQ frames and not MAX_SEQ + 1 frames may be outstanding at any instant, even though there are MAX_SEQ + 1 distinct sequence numbers: 0, 1, 2, ..., MAX_SEQ. To see why this restriction is required, consider the following scenario with MAX_SEQ = 7.



Simulation of multiple timers in software.

- The sender sends frames 0 through 7.
- A piggybacked acknowledgement for frame 7 eventually comes back to the sender.
- The sender sends another eight frames, again with sequence numbers 0 through 7.
- Now another piggybacked acknowledgement for frame 7 comes in.

The question is this: Did all eight frames belonging to the second batch arrive successfully, or did all eight get lost (counting discards following an error as lost)? In both cases the receiver would be sending frame 7 as the acknowledgement. The sender has no way of telling. For this reason the maximum number of outstanding frames must be restricted to MAX_SEQ.

Although protocol 5 does not buffer the frames arriving after an error, it does not escape the problem of buffering altogether. Since a sender may have to retransmit all the unacknowledged frames at a future time, it must hang on to all transmitted frames until it knows for sure that they have been accepted by the receiver. When an acknowledgement comes in for frame n, frames n - 1, n - 2, and so on are also automatically acknowledged. This property is especially important when some of the previous acknowledgement-bearing frames were lost or garbled. Whenever any acknowledgement comes in, the data link layer checks to see if any buffers can now be released. If buffers can be released (i.e., there is some room available in the window), a previously blocked network layer can now be allowed to cause more network_layer_ready events.

For this protocol, we assume that there is always reverse traffic on which to piggyback acknowledgements. If there is not, no acknowledgements can be sent. Protocol 4 does not need this assumption since it sends back one frame every time it receives a frame, even if it has just already sent that frame. In the next protocol we will solve the problem of one-way traffic in an elegant way.

Because protocol 5 has multiple outstanding frames, it logically needs multiple timers, one per outstanding frame. Each frame times out independently of all the other ones. All of these timers can easily be simulated in software, using a single hardware clock that causes interrupts periodically. The pending timeouts form a linked list, with each node of the list telling the number of clock ticks until the timer expires, the frame being timed, and a pointer to the next node.

Assume that the clock ticks once every 100 msec. Initially, the real time is 10:00:00.0; three timeouts are pending, at 10:00:00.5, 10:00:01.3, and 10:00:01.9. Every time the hardware clock ticks, the real time is updated and the tick counter at the head of the list is decremented. When the tick counter becomes zero, a timeout is caused and the node is removed from the list, as shown in Fig. 3-18(b). Although this organization requires the list to be scanned when start_timer or stop_timer is called, it does not require much work per tick. In

protocol 5, both of these routines have been given a parameter, indicating which frame is to be timed.

A Protocol Using Selective Repeat

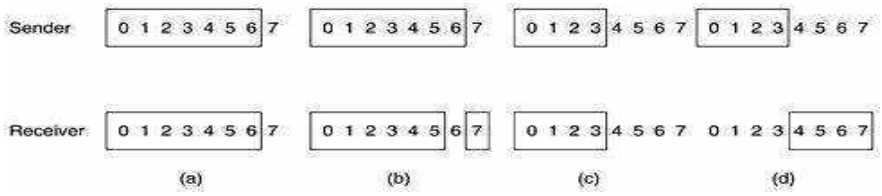
Protocol 5 works well if errors are rare, but if the line is poor, it wastes a lot of bandwidth on retransmitted frames. An alternative strategy for handling errors is to allow the receiver to accept and buffer the frames following a damaged or lost one. Such a protocol does not discard frames merely because an earlier frame was damaged or lost.

In this protocol, both sender and receiver maintain a window of acceptable sequence numbers. The sender's window size starts out at 0 and grows to some predefined maximum, MAX_SEQ. The receiver's window, in contrast, is always fixed in size and equal to MAX_SEQ. The receiver has a buffer reserved for each sequence number within its fixed window. Associated with each buffer is a bit (arrived) telling whether the buffer is full or empty. Whenever a frame arrives, its sequence number is checked by the function between to see if it falls within the window. If so and if it has not already been received, it is accepted and stored. This action is taken without regard to whether or not it contains the next packet expected by the network layer. Of course, it must be kept within the data link layer and not passed to the network layer until all the lower-numbered frames have already been delivered to the network layer in the correct order

Non-sequential receive introduces certain problems not present in protocols in which frames are only accepted in order. We can illustrate the trouble most easily with an example. Suppose that we have a 3-bit sequence number, so that the sender is permitted to transmit up to seven frames before being required to wait for an acknowledgement. The sender now transmits frames 0 through 6. The receiver's window allows it to accept any frame with sequence number between 0 and 6 inclusive. All seven frames arrive correctly, so the receiver acknowledges them and advances its window to allow receipt of 7, 0, 1, 2, 3, 4, or 5, as shown in Fig. 3-20(b). All seven buffers are marked empty.

Initial situation with a window of size seven. (b) After seven frames have been sent and received but not acknowledged. (c) Initial situation with a window size of four. (d) After four frames have been sent and received but not acknowledged.

A sliding window protocol using selective repeat



It is at this point that disaster strikes in the form of a lightning bolt hitting the telephone pole and wiping out all the acknowledgements. The sender eventually times out and retransmits frame 0. When this frame arrives at the receiver, a check is made to see if it falls within the receiver's window. Unfortunately,_frame 0 is within the new window, so it will be accepted. The receiver sends a piggybacked acknowledgement for frame 6, since 0 through 6 have been received.

The sender is happy to learn that all its transmitted frames did actually arrive correctly, so it advances its window and immediately sends frames 7, 0, 1, 2, 3, 4, and 5. Frame 7 will be accepted by the receiver and its packet will be passed directly to the network layer. Immediately thereafter, the receiving data link layer checks to see if it has a valid frame 0 already, discovers that it does, and passes the embedded packet to the network layer. Consequently, the network layer gets an incorrect packet, and the protocol fails.

The essence of the problem is that after the receiver advanced its window, the new range of valid sequence numbers overlapped the old one. Consequently, the following batch of frames might be either duplicates (if all the acknowledgements were lost) or new ones (if all the acknowledgements were received). The poor receiver has no way of distinguishing these two cases.

The way out of this dilemma lies in making sure that after the receiver has advanced its window, there is no overlap with the original window. To ensure that there is no overlap, the maximum window size should be at most half the range of the sequence numbers, as is done in For example, if 4 bits are used for sequence numbers, these will range from 0 to 15. Only eight unacknowledged frames should be outstanding at any instant. That way, if the receiver has just accepted frames 0 through 7 and advanced its window to permit acceptance of frames 8 through 15, it can unambiguously tell if subsequent frames are retransmissions (0 through 7) or new ones (8 through 15). In general, the window

size for protocol 6 will be (MAX_SEQ + 1)/2. Thus, for 3-bit sequence numbers, the window size is four.

An interesting question is: How many buffers must the receiver have? Under no conditions will it ever accept frames whose sequence numbers are below the lower edge of the window or frames whose sequence numbers are above the upper edge of the window. Consequently, the number of buffers needed is equal to the window size, not to the range of sequence numbers. In the above example of a 4-bit sequence number, eight buffers, numbered 0 through 7, are needed. When frame i arrives, it is put in buffer i mod 8. Notice that although i and (i 8) mod 8 are "competing" for the same buffer, they are never within the window at the same time, because that would imply a window size of at least 9.

For the same reason, the number of timers needed is equal to the number of buffers, not to the size of the sequence space. Effectively, a timer is associated with each buffer. When the timer runs out, the contents of the buffer are retransmitted.

In protocol 5, there is an implicit assumption that the channel is heavily loaded. When a frame arrives, no acknowledgement is sent immediately. Instead, the acknowledgement is piggybacked onto the next outgoing data frame. If the reverse traffic is light, the acknowledgement will be held up for a long period of time. If there is a lot of traffic in one direction and no traffic in the other direction, only MAX_SEQ packets are sent, and then the protocol blocks, which is why we had to assume there was always some reverse traffic.

In protocol 6 this problem is fixed. After an in-sequence data frame arrives, an auxiliary timer is started by start_ack_timer. If no reverse traffic has presented itself before this timer expires, a separate acknowledgement frame is sent. An interrupt due to the auxiliary timer is called an ack_timeout event. With this arrangement, one-directional traffic flow is now possible because the lack of reverse data frames onto which acknowledgements can be piggybacked is no longer an obstacle. Only one auxiliary timer exists, and if start_ack_timer is called while the timer is running, it is reset to a full acknowledgement timeout interval.

It is essential that the timeout associated with the auxiliary timer be appreciably shorter than the timer used for timing out data frames. This condition

is required to make sure a correctly received frame is acknowledged early enough that the frame's retransmission timer does not expire and retransmit the frame.

Protocol 6 uses a more efficient strategy than protocol 5 for dealing with errors. Whenever the receiver has reason to suspect that an error has occurred, it sends a negative acknowledgement (NAK) frame back to the sender. Such a frame is a request for retransmission of the frame specified in the NAK. There are two cases when the receiver should be suspicious: a damaged frame has arrived or a frame other than the expected one arrived (potential lost frame). To avoid making multiple requests for retransmission of the same lost frame, the receiver should keep track of whether a NAK has already been sent for a given frame. The variable no_nak in protocol 6 is true if no NAK has been sent yet for frame_expected. If the NAK gets mangled or lost, no real harm is done, since the sender will eventually time out and retransmit the missing frame anyway. If the wrong frame arrives after a NAK has been sent and lost, no_ nak will be true and the auxiliary timer will be started. When it expires, an ACK will be sent to resynchronize the sender to the receiver's current status.

In some situations, the time required for a frame to propagate to the destination, be processed there, and have the acknowledgement come back is (nearly) constant. In these situations, the sender can adjust its timer to be just slightly larger than the normal time interval expected between sending a frame and receiving its acknowledgement. However, if this time is highly variable, the sender is faced with the choice of either setting the interval to a small value (or risking unnecessary retransmissions), or setting it to a large value (and going idle for a long period after an error).

Both choices waste bandwidth. If the reverse traffic is sporadic, the time before acknowledgement will be irregular, being shorter when there is reverse traffic and longer when there is not. Variable processing time within the receiver can also be a problem here. In general, whenever the standard deviation of the acknowledgement interval is small compared to the interval itself, the timer can be set "tight" and NAKs are not useful. Otherwise the timer must be set "loose," to avoid unnecessary retransmissions, but NAKs can appreciably speed up retransmission of lost or damaged frames.

Closely related to the matter of timeouts and NAKs is the question of determining which frame caused a timeout. In protocol 5, it is always

ack_expected, because it is always the oldest. In protocol 6, there is no trivial way to determine who timed out. Suppose that frames 0 through 4 have been transmitted, meaning that the list of outstanding frames is 01234, in order from oldest to youngest. Now imagine that 0 times out, 5 (a new frame) is transmitted, 1 times out, 2 times out, and 6 (another new frame) is transmitted. At this point the list of outstanding frames is 3405126, from oldest to youngest. If all inbound traffic (i.e., acknowledgement-bearing frames) is lost for a while, the seven outstanding frames will time out in that order.

To keep the example from getting even more complicated than it already is, we have not shown the timer administration. Instead, we just assume that the variable oldest_frame is set upon timeout to indicate which frame timed out.

The Medium Access Control Sub-layer

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

The protocols used to determine who goes next on a multi-access channel belong to a sub-layer of the data link layer called the **MAC** (**Medium Access Control**) sub-layer. The MAC sub-layer is especially important in LANs, many of which use a multi-access channel as the basis for communication. WANs, in contrast, use point-to-point links, except for satellite networks. Because multi-access channels and LANs are so closely related, in this chapter we will discuss LANs in general, including a few issues that are not strictly part of the MAC sub-layer.

The Channel Allocation Problem

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

Static Channel Allocation in LANs and MANs

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

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

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

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

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

rate is μC frames/sec. From queueing theory it can be shown that for Poisson arrival and service times,

For example, if C is 100 Mbps, the mean frame length, 1/μ, is 10,000 bits, and the frame arrival rate, λ, is 5000 frames/sec, then T = 200 μsec. Note that if we ignored the queueing delay and just asked how long it takes to send a 10,000 bit frame on a 100-Mbps network, we would get the (incorrect) answer of 100 μsec. That result only holds when there is no contention for the channel.

Now let us divide the single channel into N independent sub-channels, each with capacity C/N bps. The mean input rate on each of the sub-channels will now be λ/N. Re-computing T we get

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

Equation 4

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

Precisely the same arguments that apply to FDM also apply to time division multiplexing (TDM). Each user is statically allocated every the time slot. If a user does not use the allocated slot, it just lies fallow. The same holds if we split up the networks physically. Using our previous example again, if we were to replace the 100-Mbps network with 10 networks of 10 Mbps each and statically allocate each user to one of them, the mean delay would jump from 200 μsec to 2 msec.

Since none of the traditional static channel allocation methods work well with bursty traffic, we will now explore dynamic methods.

Dynamic Channel Allocation in LANs and MANs

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

Station Model:

The model consists of independent **stations** (e.g., computers, telephones, or personal communicators), each with a program or user that generates frames for transmission. Stations are sometimes called **terminals**. The probability of a

frame being generated in an interval of length Δt is λΔt, where λ is a constant (the arrival rate of new frames). Once a frame has been generated, the station is blocked and does nothing until the frame has been successfully transmitted.

Single Channel Assumption:

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

Collision Assumption:

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

Continuous Time:

Frame transmission can begin at any instant. There is no master clock dividing time into discrete intervals.

Slotted Time:

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

Carrier Sense:

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

No Carrier Sense:

Stations cannot sense the channel before trying to use it. They just go ahead and transmit. Only later can they determine whether the transmission was successful.

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

multi-programmed stations that can generate work while a station is blocked, but the analysis of these stations is much more complex.

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

The collision assumption is also basic, although in some systems (notably spread spectrum), this assumption is relaxed, with surprising results. Also, some LANs, such as token rings, pass a special token from station to station, possession of which allows the current holder to transmit a frame. But in the coming sections we will stick to the single channel with contention and collisions model.

Two alternative assumptions about time are possible. Either it is continuous (4a) or it is slotted (4b). Some systems use one and some systems use the other, so we will discuss and analyze both. For a given system, only one of them holds.

Similarly, a network can either have carrier sensing (5a) or not have it (5b). LANs generally have carrier sense. However, wireless networks cannot use it effectively because not every station may be within radio range of every other station. Stations on wired carrier sense networks can terminate their transmission prematurely if they discover that it is colliding with another transmission. Collision detection is rarely done on wireless networks, for engineering reasons. Note that the word "carrier" in this sense refers to an electrical signal on the cable and has nothing to do with the common carriers (e.g., telephone companies) that date back to the Pony Express days.

Multiple Access Protocols

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

ALOHA

In the 1970s, Norman Abramson and his colleagues at the University of Hawaii devised a new and elegant method to solve the channel allocation problem. Their work has been extended by many researchers since then (Abramson, 1985). Although Abramson's work, called the ALOHA system, used

ground-based radio broadcasting, the basic idea is applicable to any system in which uncoordinated users are competing for the use of a single shared channel.

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

In pure ALOHA, frames are transmitted at completely arbitrary times.

Pure ALOHA

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

We have made the frames all the same length because the throughput of ALOHA systems is maximized by having a uniform frame size rather than by allowing variable length frames.

Whenever two frames try to occupy the channel at the same time, there will be a collision and both will be garbled. If the first bit of a new frame overlaps with just the last bit of a frame almost finished, both frames will be totally

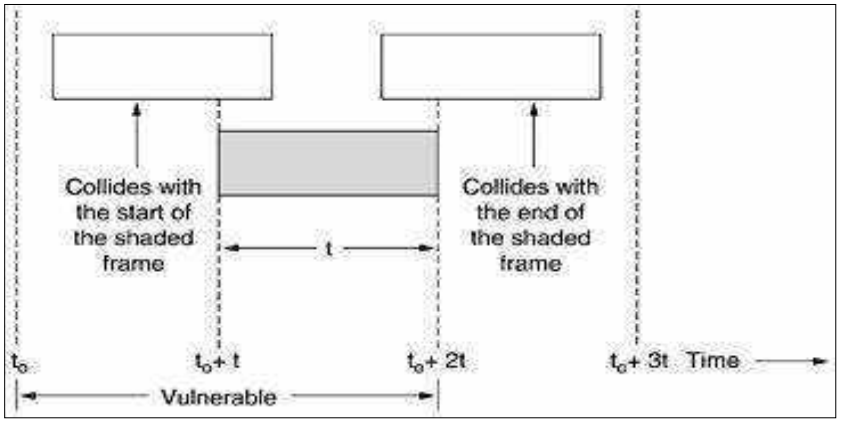
destroyed and both will have to be retransmitted later. The checksum cannot (and should not) distinguish between a total loss and a near miss. Bad is bad.

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

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

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

Vulnerable period for the shaded frame



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

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

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

Equation 4

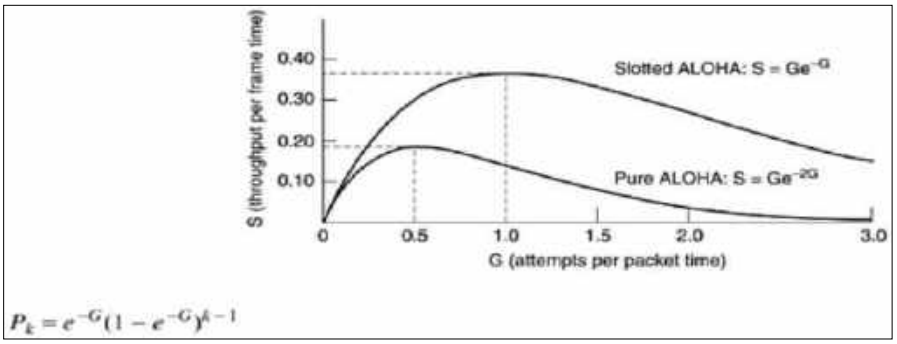
So the probability of zero frames is just e^{-G}. In an interval two frame times long, the mean number of frames generated is 2G. The probability of no other traffic being initiated during the entire vulnerable period is thus given by $P_0 = e^{-2G}$. Using $S = GP_0$, we get

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

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

Slotted ALOHA

Throughput versus offered traffic for ALOHA systems



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

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

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

Equation 4

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

transmission requiring exactly k attempts, (i.e., k - 1 collisions followed by one success) is the expected number of transmissions, E, per carriage return typed is then

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

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

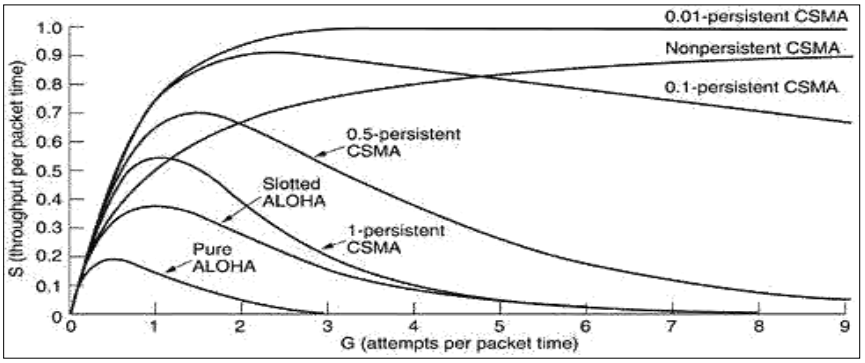
Slotted Aloha is important for a reason that may not be initially obvious. It was devised in the 1970s, used in a few early experimental systems, then almost forgotten. When Internet access over the cable was invented, all of a sudden there was a problem of how to allocate a shared channel among multiple competing users, and slotted Aloha was pulled out of the garbage can to save the day. It has often happened that protocols that are perfectly valid fall into disuse for political reasons (e.g., some big company wants everyone to do things its way), but years later some clever person realizes that a long-discarded protocol solves his current problem. For this reason, in this chapter we will study a number of elegant protocols that are not currently in widespread use, but might easily be used in future applications, provided that enough network designers are aware of them. Of course, we will also study many protocols that are in current use as well.

Carrier Sense Multiple Access Protocols

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

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

Comparison of the channel utilization versus load for various random access protocols



Persistent and Non-persistent CSMA

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

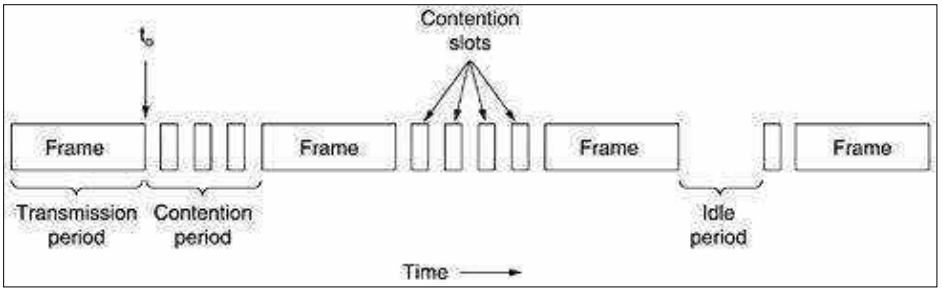
The propagation delay has an important effect on the performance of the protocol. There is a small chance that just after a station begins sending, another station will become ready to send and sense the channel. If the first station's signal has not yet reached the second one, the latter will sense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

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

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

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

CSMA/CD can be in one of three states: contention, transmission, or idle



CSMA with Collision Detection

Persistent and non-persistent CSMA protocols are clearly an improvement over ALOHA because they ensure that no station begins to transmit when it senses the channel busy. Another improvement is for stations to abort their transmissions as soon as they detect a collision. In other words, if two stations sense the channel to be idle and begin transmitting simultaneously, they will both detect the collision almost immediately. Rather than finish transmitting their frames, which are irretrievably garbled anyway, they should abruptly stop

transmitting as soon as the collision is detected. Quickly terminating damaged frames saves time and bandwidth. This protocol, known as **CSMA/CD** (**CSMA with Collision Detection**) is widely used on LANs in the MAC sub-layer. In particular, it is the basis of the popular Ethernet LAN, so it is worth devoting some time to looking at it in detail.

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

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

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

Based on this reasoning, you might think that a station not hearing a collision for a time equal to the full cable propagation time after starting its transmission could be sure it had seized the cable. By "seized," we mean that all other stations knew it was transmitting and would not interfere. This conclusion is wrong. Consider the following worst-case scenario. Let the time for a signal to propagate between the two farthest stations be τ. At t_0, one station begins transmitting. At $\tau - \varepsilon$, an instant before the signal arrives at the most distant station, that station also begins transmitting. Of course, it detects the collision almost instantly and stops, but the little noise burst caused by the collision does not get back to the original station until time $2\tau - \varepsilon$. In other words, in the worst case a station cannot be sure that it has seized the channel until it has transmitted for 2τ without hearing a collision. For this reason we will model the contention

interval as a slotted ALOHA system with slot width 2τ. On a 1-km long coaxial cable, τ ≅ μsec. For simplicity we will assume that each slot contains just 1 bit. Once the channel has been seized, a station can transmit at any rate it wants to, of course, not just at 1 bit per 2τ sec.

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

It is also worth noting that a sending station must continually monitor the channel, listening for noise bursts that might indicate a collision. For this reason, CSMA/CD with a single channel is inherently a half-duplex system. It is impossible for a station to transmit and receive frames at the same time because the receiving logic is in use, looking for collisions during every transmission.

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

Collision-Free Protocols

Although collisions do not occur with CSMA/CD once a station has unambiguously captured the channel, they can still occur during the contention period. These collisions adversely affect the system performance, especially when the cable is long (i.e., large τ) and the frames are short. And CSMA/CD is not universally applicable. In this section, we will examine some protocols that resolve the contention for the channel without any collisions at all, not even during the contention period. Most of these are not currently used in major systems, but in a rapidly changing field, having some protocols with excellent properties available for future systems is often a good thing.

The binary countdown protocol. A dash indicates silence

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

propagation delay is negligible. The basic question remains: Which station gets the channel after a successful transmission.

A Bit-Map Protocol

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

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

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

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

The prospects for high-numbered stations are brighter. Generally, these will only have to wait half a scan (N/2 bit slots) before starting to transmit. High-numbered stations rarely have to wait for the next scan. Since low-numbered stations must wait on average 1.5N slots and high-numbered stations must wait on average 0.5N slots, the mean for all stations is N slots. The channel efficiency

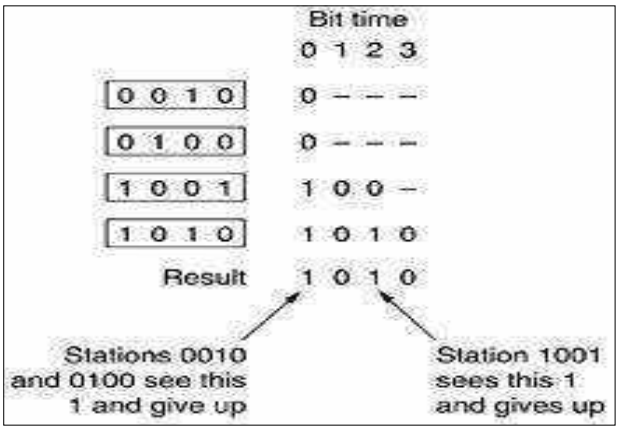
at low load is easy to compute. The overhead per frame is N bits, and the amount of data is d bits, for an efficiency of d/(N + d).

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

Binary Countdown

A problem with the basic bit-map protocol is that the overhead is 1 bit per station, so it does not scale well to networks with thousands of stations. We can do better than that by using binary station addresses. A station wanting to use the channel now broadcasts its address as a binary bit string, starting with the high-order bit. All addresses are assumed to be the same length. The bits in each address position from different stations are BOOLEAN ORed together. We will call this protocol **binary countdown**. It was used in Datakit (Fraser, 1987). It implicitly assumes that the transmission delays are negligible so that all stations see asserted bits essentially instantaneously.

The binary countdown protocol
A dash indicates silence



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

1 and know that a higher-numbered station is competing for the channel, so they give up for the current round. Stations 1001 and 1010 continue.

The next bit is 0, and both stations continue. The next bit is 1, so station 1001 gives up. The winner is station 1010 because it has the highest address. After winning the bidding, it may now transmit a frame, after which another bidding cycle starts. The protocol is illustrated in Fig. 4-7. It has the property that higher-numbered stations have a higher priority than lower-numbered stations, which may be either good or bad, depending on the context.

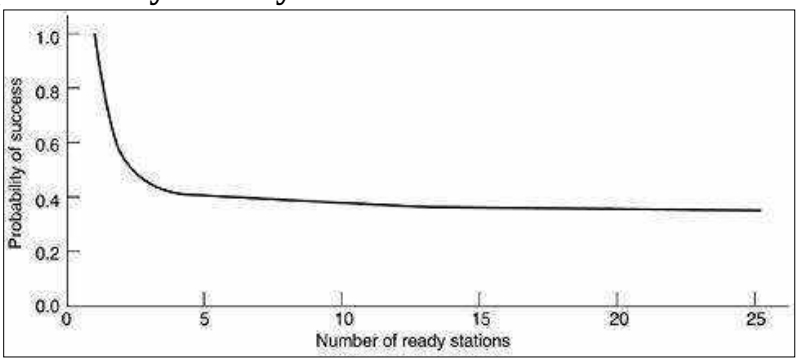
The channel efficiency of this method is $d/(d + \log_2 N)$. If, however, the frame format has been cleverly chosen so that the sender's address is the first field in the frame, even these $\log_2 N$ bits are not wasted, and the efficiency is 100 percent.

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

Binary countdown is an example of a simple, elegant, and efficient protocol that is waiting to be rediscovered. Hopefully, it will find a new home someday.

Limited-Contention Protocols

Acquisition probability for a symmetric contention channel



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

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

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

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

Equation 4

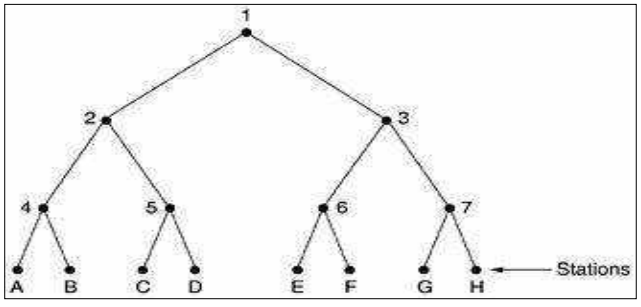
Probability is plotted in for small numbers of stations, the chances of success are good, but as soon as the number of stations reaches even five, the probability has dropped close to its asymptotic value of 1/e.

It is fairly obvious that the probability of some station acquiring the channel can be increased only by decreasing the amount of competition. The limited-contention protocols do precisely that. They first divide the stations into (not necessarily disjoint) groups. Only the members of group 0 are permitted to compete for slot 0. If one of them succeeds, it acquires the channel and transmits its frame. If the slot lies fallow or if there is a collision, the members of group 1 contend for slot 1, etc. By making an appropriate division of stations into groups, the amount of contention for each slot can be reduced, thus operating each slot near the left end of the trick is how to assign stations to slots. Before looking at the general case, let us consider some special cases. At one extreme, each group has but one member. Such an assignment guarantees that there will never be collisions because at most one station is contending for any given slot. We have seen such protocols before (e.g., binary countdown). The next special case is to assign two stations per group. The probability that both will try to transmit during a slot is p^2, which for small p is negligible. As more and more stations are assigned to the same slot,the probability of a collision grows, but the length of the bit-map scan needed to give everyone a chance shrinks. The limiting case is a single group containing all stations (i.e., slotted ALOHA). What we need is a way to assign stations to slots dynamically, with many stations per slot when the load is low and few (or even just one) station per slot when the load is high.

The tree for eight stations.

The Adaptive Tree Walk Protocol

The tree for eight stations



One particularly simple way of performing the necessary assignment is to use the algorithm devised by the U.S. Army for testing soldiers for syphilis during World War II (Dorfman, 1943). In short, the Army took a blood sample from N

soldiers. A portion of each sample was poured into a single test tube. This mixed sample was then tested for antibodies. If none were found, all the soldiers in the group were declared healthy. If antibodies were present, two new mixed samples were prepared, one from soldiers 1 through N/2 and one from the rest. The process was repeated recursively until the infected soldiers were determined.

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

In essence, if a collision occurs during slot 0, the entire tree is searched, depth first, to locate already stations. Each bit slot is associated with some particular node in the tree. If a collision occurs, the search continues recursively with the node's left and right children. If a bit slot is idle or if only one station transmits in it, the searching of its node can stop because all ready stations have been located. (Were there more than one, there would have been a collision.)

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

To proceed, let us number the levels of the tree from the top, with node 1 at level 0, nodes 2 and 3 at level 1, etc. Notice that each node at level i has a fraction 2^{-i} of the stations below it. If the q ready stations are uniformly distributed, the expected number of them below a specific node at level i is just $2^{-i}q$. Intuitively, we would expect the optimal level to begin searching the tree as

the one at which the mean number of contending stations per slot is 1, that is, the level at which $2^{-i}q = 1$. Solving this equation, we find that $i = \log_2 q$.

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

Wavelength Division Multiple Access Protocols

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

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

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

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

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

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

- A fixed-wavelength receiver for listening to its own control channel.
- A tunable transmitter for sending on other stations' control channels.
- A fixed-wavelength transmitter for outputting data frames.
- A tunable receiver for selecting a data transmitter to listen to.

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

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

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

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

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

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

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

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

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

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

Numerous other WDMA protocols have been proposed and implemented, differing in various details. Some have only one control channel; others have

multiple control channels. Some take propagation delay into account; others do not. Some make tuning time an explicit part of the model; others ignore it. The protocols also differ in terms of processing complexity, throughput, and scalability. When a large number of frequencies are being used, the system is sometimes called **DWDM** (**Dense Wavelength Division Multiplexing**). For more information see (Bogineni et al., 1993; Chen, 1994; Goralski, 2001; Kartalopoulos, 1999; and Levine and Akyildiz, 1995).

Wireless LAN Protocols

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

A wireless LAN (a) A transmitting. (b) B transmitting

To achieve true mobility, notebook computers need to use radio (or infrared) signals for communication. In this manner, dedicated users can read and send e-mail while hiking or boating. A system of notebook computers that communicate by radio can be regarded as a wireless LAN, These LANs have somewhat different properties than conventional LANs and require special MAC sub-layer protocols. In this section we will examine some of these protocols. More information about wireless LANs can be found in (Geier, 2002; and O'Hara and Petrick, 1999). A common configuration for a wireless LAN is an office building with base stations (also called access points) strategically placed around the building. All the base stations are wired together using copper or fiber. If the transmission power of the base stations and notebooks is adjusted to have a range of 3 or 4 meters, then each room becomes a single cell and the entire building becomes a large cellular system, as in the traditional cellular telephony systems we studied in Chap. 2. Unlike cellular telephone systems, each cell has only one

channel, covering the entireavailable bandwidth and covering all the stations in its cell. Typically, its bandwidth is 11 to 54 Mbps.

In our discussions below, we will make the simplifying assumption that all radio transmitters have some fixed range. When a receiver is within range of two active transmitters, the resulting signal will generally be garbled and useless, in other words, we will not consider CDMA-type systems further in this discussion. It is important to realize that in some wireless LANs, not all stations are within range of one another, which leads to a variety of complications. Furthermore, for indoor wireless LANs, the presence of walls between stations can have a major impact on the effective range of each station.

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

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

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

The problem is that before starting a transmission, a station really wants to know whether there is activity around the receiver. CSMA merely tells it whether there is activity around the station sensing the carrier. With a wire, all signals

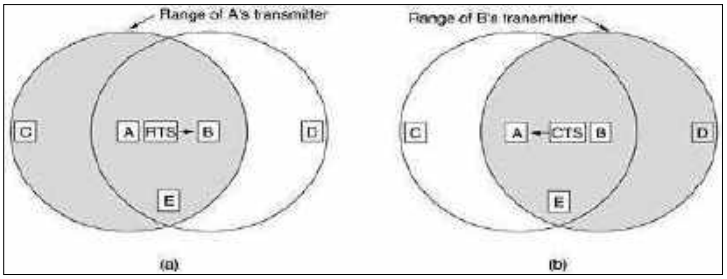
propagate to all stations so only one transmission can take place at once anywhere in the system. In a system based on short-range radio waves, multiple transmissions can occur simultaneously if they all have different destinations and these destinations are out of range of one another.

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

MACA and MACAW

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

The MACA protocol (a) A sending an RTS to B (b) B responding with a CTS to A



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

Now let us see how stations overhearing either of these frames react. Any station hearing the RTS is clearly close to A and must remain silent long enough for the CTS to be transmitted back to A without conflict. Any station hearing the

CTS is clearly close to B and must remain silent during the upcoming data transmission, whose length it can tell by examining the CTS frame.

In Fig. 4-12, C is within range of A but not within range of B. Therefore, it hears the RTS from A but not the CTS from B. As long as it does not interfere with the CTS, it is free to transmit while the data frame is being sent. In contrast, D is within range of B but not A. It does not hear the RTS but does hear the CTS. Hearing the CTS tips it off that it is close to a station that is about to receive a frame, so it defers sending anything until that frame is expected to be finished. Station E hears both control messages and, like D, must be silent until the data frame is complete.

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

Based on simulation studies of MACA, Bharghavan et al. (1994) fine tuned MACA to improve its performance and renamed their new protocol **MACAW** (**MACA for Wireless**).

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

Ethernet

We have now finished our general discussion of channel allocation protocols in the abstract, so it is time to see how these principles apply to real systems, in particular, LANs., the IEEE has standardized a number of local area networks and metropolitan areanetworks under the name of IEEE 802. A few have survived but many have not. Some people who believe in reincarnation think that Charles Darwin came back as amember of the IEEE Standards Association to weed out the unfit. The most important of the survivors are 802.3 (Ethernet) and 802.11 (wireless LAN). With 802.15 (Bluetooth) and

802.16 (wireless MAN), it is too early to tell. Please consult the 5th edition of this book to find out. Both 802.3 and 802.11 have different physical layers and different MAC sub-layers but converge on the same logical link control sub-layer (defined in 802.2), so they have the same interface to the network layer.

Ethernet Cabling

Since the name "Ethernet" refers to the cable (the ether), let us start our discussion there.
Four types of cabling are commonly used.

The most common kinds of Ethernet cabling

Name	Cable	Max. seg.	Nodes/seg.	Advantages
10Base5	Thick coax	500 m	100	Original cable; now obsolete
10Base2	Thin coax	185 m	30	No hub needed
10Base-T	Twisted pair	100 m	1024	Cheapest system
10Base-F	Fiber optics	2000 m	1024	Best between buildings

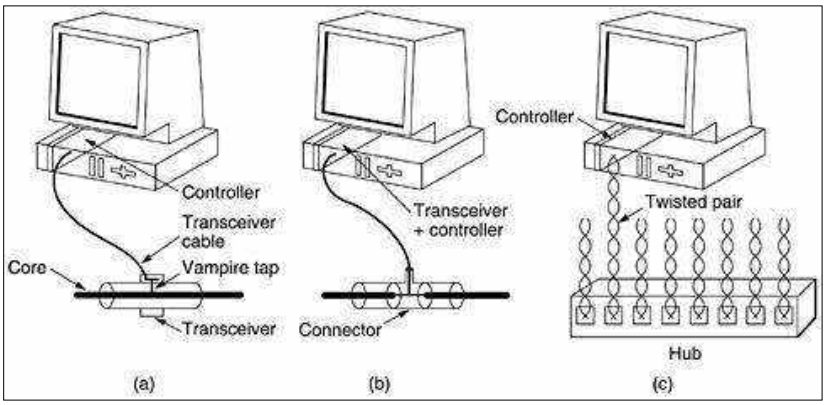
Historically, **10Base5** cabling, popularly called **thick Ethernet,** came first. It resembles a yellow garden hose, with markings every 2.5 meters to show where the taps go. (The 802.3 standard does not actually require the cable to be yellow, but it does suggest it.) Connections to it are generally made using **vampire taps**, in which a pin is very carefully forced halfway into the coaxial cable's core. The notation 10Base5 means that it operates at 10 Mbps, uses baseband signaling, and can support segments of up to 500 meters. The first number is the speed in Mbps. Then comes the word "Base" (or sometimes "BASE") to indicate baseband transmission. There used to be a broadband variant, 10Broad36, but it never

caught on in the marketplace and has since vanished. Finally, if the medium is coax, its length is given rounded to units of 100 m after "Base."

Historically, the second cable type was **10Base2**, or **thin Ethernet,** which, in contrast to the garden-hose-like thick Ethernet, bends easily. Connections to it are made using industry-standard BNC connectors to form T junctions, rather than using vampire taps. BNC connectors are easier to use and more reliable. Thin Ethernet is much cheaper and easier to install, but it can run for only 185 meters per segment, each of which can handle only 30 machines.

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

Three kinds of Ethernet cabling
(a) 10Base5. (b) 10Base2. (c) 10Base-T



With 10Base5, a **transceiver cable** or **drop cable** connects the transceiver to an interface board in the computer. The transceiver cable may be up to 50 meters long and contains five individually shielded twisted pairs. Two of the pairs are for data in and data out, respectively.

Two more are for control signals in and out. The fifth pair, which is not always used, allows the computer to power the transceiver electronics. Some transceivers allow up to eight nearby computers to be attached to them, to reduce the number of transceivers needed.

The transceiver cable terminates on an interface board inside the computer. The interface board contains a controller chip that transmits frames to, and receives frames from, the transceiver. The controller is responsible for assembling the data into the proper frame format, as well as computing checksums on outgoing frames and verifying them on incoming frames.

Some controller chips also manage a pool of buffers for incoming frames, a queue of buffers to be transmitted, direct memory transfers with the host computers, and other aspects of network management.

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

With 10Base-T, there is no shared cable at all, just the hub (a box full of electronics) to which each station is connected by a dedicated (i.e., not shared) cable. Adding or removing a station is simpler in this configuration, and cable breaks can be detected easily. The disadvantage of 10Base-T is that the maximum cable run from the hub is only 100 meters, maybe 200 meters if very high quality category 5 twisted pairs are used. Nevertheless, 10Base-T quickly became dominant due to its use of existing wiring and the ease of maintenance that it offers. A faster version of 10Base-T (100Base-T) will be discussed later in this chapter.

A fourth cabling option for Ethernet is **10Base-F**, which uses fiber optics. This alternative is expensive due to the cost of the connectors and terminators, but it has excellent noise immunity and is the method of choice when running between buildings or widely-separated hubs. Runs of up to km are allowed. It also offers good security since wiretapping fiber is much more difficult than wiretapping copper wire.

Manchester Encoding

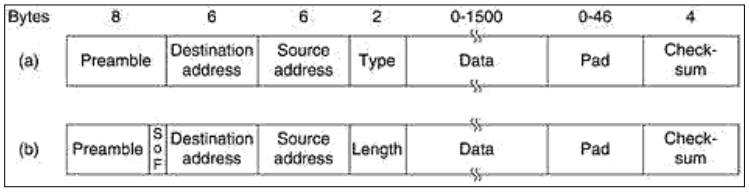
None of the versions of Ethernet uses straight binary encoding with 0 volts for a 0 bit and 5 volts for a 1 bit because it leads to ambiguities. If one station sends the bit string 0001000, others might falsely interpret it as 10000000 or 01000000 because they cannot tell the difference between an idle sender (0 volts) and a 0 bit (0 volts). This problem can be solved by using +1 volts for a 1 and -1 volts for a 0, but there is still the problem of a receiver sampling the signal at a

slightly different frequency than the sender used to generate it. Different clock speeds can cause the receiver and sender to get out of synchronization about where the bit boundaries are, especially after a long run of consecutive 0s or a long run of consecutive 1s.

The Ethernet MAC Sub-layer Protocol

Each frame starts with a Preamble of 8 bytes, each containing the bit pattern 10101010. The Manchester encoding of this pattern produces a 10-MHz square wave for 6.4 μsec to allow the receiver's clock to synchronize with the sender's. They are required to stay synchronized for the rest of the frame, using the Manchester encoding to keep track of the bit boundaries.

Frame formats. (a) DIX Ethernet. (b) IEEE 802.3



The frame contains two addresses, one for the destination and one for the source. The standard allows 2-byte and 6-byte addresses, but the parameters defined for the 10-Mbps baseband standard use only the 6-byte addresses. The high-order bit of the destination address is a 0 for ordinary addresses and 1 for group addresses. Group addresses allow multiple stations to listen to a single address. When a frame is sent to a group address, all the stations in the group receive it. Sending to a group of stations is called **multicast**. The address consisting of all 1 bits is reserved for **broadcast**. A frame containing all 1s in the destination field is accepted by all stations on the network. The difference between multicast and broadcast is important enough to warrant repeating. A multicast frame is sent to a selected group of stations on the Ethernet; a broadcast frame is sent to all stations on the Ethernet. Multicast is more selective, but involves group management. Broadcasting is coarser but does not require any group management.

The Binary Exponential Back off Algorithm

Let us now see how randomization is done when a collision occurs. After a collision, time is divided into discrete slots whose length is equal to the worst-case round-trip propagation time on the ether (2τ). To accommodate the longest

path allowed by Ethernet, the slot time has been set to 512 bit times, or 51.2 μsec as mentioned above.

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

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

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

Ethernet Performance

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

Equation 4

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

Since each slot has a duration 2τ, the mean contention interval, w, is 2τ/A. Assuming optimal p, the mean number of contention slots is never more than e, so w is at most 2τe ~~=~~ 5.4τ. If the mean frame takes P sec to transmit, when many stations have frames to send, $\text{Channel efficiency} = \frac{P}{P + 2\tau/A}$

Equation 4

Here we see where the maximum cable distance between any two stations enters into the performance figures, giving rise to topologies. The longer the cable, the longer the contention interval. This observation is why the Ethernet standard specifies a maximum cable length.

S

It is instructive to formulate Eq. (4-6) in terms of the frame length, F, the network bandwidth, B, the cable length, L, and the speed of signal propagation, c, for the optimal case of e contention slots per frame. With P = F/B, Eq. (4-6) becomes

Equation 4

$$\text{Channel efficiency} = \frac{1}{1 + 2BLe/cF}$$

Switched Ethernet

As more and more stations are added to an Ethernet, the traffic will go up. Eventually, the LAN will saturate. One way out is to go to a higher speed, say, from 10 Mbps to 100 Mbps. But with the growth of multimedia, even a 100-Mbps or 1-Gbps Ethernet can become saturated.

Fortunately, there is an additional way to deal with increased load: switched Ethernet. The heart of this system is a **switch** containing a high-speed backplane and room for typically 4 to 32 plug-in line cards, each containing one to eight connectors. Most often, each connector has a 10Base-T twisted pair connection to a single host computer.

Fast Ethernet

At first, 10 Mbps seemed like heaven, just as 1200-bps modems seemed like heaven to the early users of 300-bps acoustic modems. But the novelty wore off quickly. As a kind of corollary to Parkinson's Law ("Work expands to fill the time available for its completion"), it seemed that data expanded to fill the bandwidth available for their transmission. To pump up the speed, various industry groups proposed two new ring-based optical LANs. One was called **FDDI** (**Fiber Distributed Data Interface**) and the other was called **Fibre Channel.** To make a long story short, while both were used as backbone networks, neither one made the breakthrough to the desktop. In both cases, the station management was too complicated, which led to complex chips and high prices. The lesson that should have been learned here was KISS (Keep It Simple, Stupid).

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

Gigabit Ethernet

The ink was barely dry on the fast Ethernet standard when the 802 committee began working on a yet faster Ethernet (1995). It was quickly dubbed **gigabit Ethernet** and was ratified by IEEE in 1998 under the name 802.3z. This identifier suggests that gigabit Ethernet is going to be the end of the line unless somebody quickly invents a new letter after z. Below we will discuss some of the key features of gigabit Ethernet. More information can be found in (Seifert, 1998).

The 802.3z committee's goals were essentially the same as the 802.3u committee's goals: make Ethernet go 10 times faster yet remain backward compatible with all existing Ethernet standards. In particular, gigabit Ethernet had to offer unacknowledged datagram service with both unicast and multicast, use the same 48-bit addressing scheme already in use, and maintain the same frame format, including the minimum and maximum frame sizes. The final standard met all these goals.

All configurations of gigabit Ethernet are point-to-point rather than multi-drop as in the original 10 Mbps standard, now honored as **classic Ethernet**. In the simplest gigabit Ethernet configuration, two computers are directly connected to each other. The more common case, however, is having a switch or a hub connected to multiple computers and possibly additional switches or hubs, In both configurations each individual Ethernet cable has exactly two devices on it, no more and no fewer.

Wireless LANs

Although Ethernet is widely used, it is about to get some competition. Wireless LANs are increasingly popular, and more and more office buildings, airports, and other public places are being outfitted with them. Wireless LANs can operate in one of two configurations, with a base station and without a base station. Consequently, the 802.11 LAN standard takes this into account and makes provision for both arrangements, as we will see shortly.

We gave some background information on 802.11 in Sec. 1.5.4. Now is the time to take a closer look at the technology. In the following sections we will look at the protocol stack, physical layer radio transmission techniques, MAC sub-layer protocol, frame structure, and services. For more information about 802.11, see (Crow et al., 1997; Geier, 2002; Heegard et al., 2001; Kapp, 2002; O'Hara and Petrick, 1999; and Severance, 1999). To hear the truth from the mouth of the horse, consult the published 802.11 standard itself.

The 802.11 Protocol Stack

The protocols used by all the 802 variants, including Ethernet, have a certain commonality of structure. A partial view of the 802.11 protocol stack] . The physical layer corresponds to the OSI physical layer fairly well, but the data link layer in all the 802 protocols is split into two or more sub-layers. In 802.11, the MAC (Medium Access Control) sub-layer determines how the channel is allocated, that is, who gets to transmit next. Above it is the LLC (Logical Link Control) sub-layer, whose job it is to hide the differences between the different 802 variants and make them indistinguishable as far as the network layer is concerned. We studied the LLC when examining Ethernet earlier in this chapter and will not repeat that material here.

The 1997 802.11 standard specifies three transmission techniques allowed in the physical layer. The infrared method uses much the same technology as television remote controls do. The other two use short-range radio, using techniques called FHSS and DSSS. Both of these use a part of the spectrum that does not require licensing (the 2.4-GHz ISM band). Radio-controlled garage door openers also use this piece of the spectrum, so your notebook computer may find itself in competition with your garage door. Cordless telephones and microwave ovens also use this band. All of these techniques operate at 1 or 2 Mbps and at low enough power that they do not conflict too much. In 1999, two new techniques were introduced to achieve higher bandwidth. These are called OFDM and HR-DSSS. They operate at up to 54 Mbps and 11 Mbps, respectively. In 2001, a second OFDM modulation was introduced, but in a different frequency band from the first one.

The 802.11 Physical Layer

Each of the five permitted transmission techniques makes it possible to send a MAC frame from one station to another. They differ, however, in the technology used and speeds achievable. A detailed discussion of these technologies is far beyond the scope of this book, but a few words on each one, along with some of the key words, may provide interested readers with terms to search for on the Internet or elsewhere for more information.

The infrared option uses diffused (i.e., not line of sight) transmission at 0.85 or 0.95 microns. Two speeds are permitted: 1 Mbps and 2 Mbps. At 1 Mbps, an encoding scheme is used in which a group of 4 bits is encoded as a 16-bit code word containing fifteen 0s and a single 1, using what is called **Gray code**. This code has the property that a small error in time synchronization leads to only a single bit error in the output. At 2 Mbps, the encoding takes 2 bits and produces a 4-bit code word, also with only a single 1, that is one of 0001, 0010, 0100, or 1000. Infrared signals cannot penetrate walls, so cells in different rooms are well isolated from each other. Nevertheless, due to the low bandwidth (and the fact that sunlight swamps infrared signals), this is not a popular option.

FHSS (Frequency Hopping Spread Spectrum)

Uses 79 channels, each 1-MHz wide, starting at the low end of the 2.4-GHz ISM band. A pseudorandom number generator is used to produce the sequence of frequencies hopped to. As long as all stations use the same seed to the pseudorandom number generator and stay synchronized in time, they will hop to

the same frequencies simultaneously. The amount of time spent at each frequency, the **dwell time**, is an adjustable parameter, but must be less than 400 msec. FHSS' randomization provides a fair way to allocate spectrum in the unregulated ISM band. It also provides a modicum of security since an intruder who does not know the hopping sequence or dwell time cannot eavesdrop on transmissions. Over longer distances, multipath fading can be an issue, and FHSS offers good resistance to it. It is also relatively insensitive to radio interference, which makes it popular for building-to-building links. Its main disadvantage is its low bandwidth.

The third modulation method, **DSSS** (**Direct Sequence Spread Spectrum**), is also restricted to 1 or 2 Mbps. The scheme used has some similarities to the CDMA system we examined in Sec. 2.6.2, but differs in other ways. Each bit is transmitted as 11 chips, using what is called a**Barker sequence**. It uses phase shift modulation at 1 Mbaud, transmitting 1 bit per baudwhen operating at 1 Mbps and 2 bits per baud when operating at 2 Mbps. For years, the FCC required all wireless communications equipment operating in the ISM bands in the U.S. to use spread spectrum, but in May 2002, that rule was dropped as new technologies emerged.

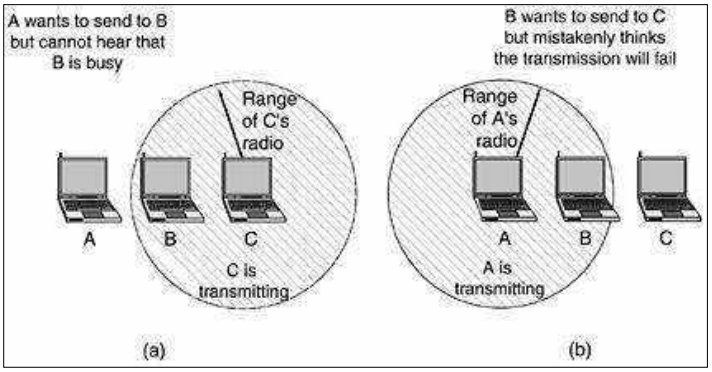
The first of the high-speed wireless LANs, **802.11a**, uses **OFDM** (**Orthogonal FrequencyDivision Multiplexing**) to deliver up to 54 Mbps in the wider 5-GHz ISM band. As the termFDM suggests, different frequencies are used—52 of them, 48 for data and 4 for synchronization—not unlike ADSL. Since transmissions are present on multiple frequencies at the same time, this technique is considered a form of spread spectrum, but different from both CDMA and FHSS. Splitting the signal into many narrow bands has some key advantages over using a single wide band, including better immunity to narrowband interference and the possibility of using noncontiguous bands. A complex encoding system is used, based on phase-shift modulation for speeds up to 18 Mbps and on QAM above that. At 54 Mbps, 216 data bits are encoded into 288-bit symbols. Part of the motivation for OFDM is compatibility with the European HiperLAN/2 system (Doufexi et al., 2002). The technique has a good spectrum efficiency in terms of bits/Hz and good immunity to multipath fading.

Next, we come to **HR-DSSS** (**High Rate Direct Sequence Spread Spectrum**), another spread spectrum technique, which uses 11 million chips/sec to achieve 11 Mbps in the 2.4-GHz band. It is called **802.11b** but is not a follow-

up to 802.11a. In fact, its standard was approved first and it got to market first. Data rates supported by 802.11b are 1, 2, 5.5, and 11 Mbps. The two slow rates run at 1 Mbaud, with 1 and 2 bits per baud, respectively, using phase shift modulation (for compatibility with DSSS). The two faster rates run at 1.375 Mbaud, with 4 and 8 bits per baud, respectively, using **Walsh/Hadamard** codes. The data rate may be dynamically adapted during operation to achieve the optimum speed possible under current conditions of load and noise. In practice, the operating speed of 802.11b is nearly always 11 Mbps. Although 802.11b is slower than 802.11a, its range is about 7 times greater, which is more important in many situations.

The 802.11 MAC Sub-layer Protocol

(a) The hidden station problem.
(b) The exposed station problem



Let us now return from the land of electrical engineering to the land of computer science. The 802.11 MAC sub-layer protocol is quite different from that of Ethernet due to the inherent complexity of the wireless environment compared to that of a wired system. With Ethernet, a station just waits until the ether goes silent and starts transmitting. If it does not receive a noise burst back within the first 64 bytes, the frame has almost assuredly been delivered correctly. With wireless, this situation does not hold.

Since not all stations are within radio range of each other, transmissions going on in one part of a cell may not be received elsewhere in the same cell. In this example, station C is transmitting to station B. If A senses the channel, it will not hear anything and falsely conclude that it may now start transmitting to B.

In addition, there is the inverse problem, the exposed station problem, illustrated in Fig. 4-26(b). Here B wants to send toCso it listens to the channel. When it hears a transmission, it falsely concludes that it may not send to C, even though A may be transmitting to D (not shown). In addition, most radios are half duplex, meaning that they cannot transmit and listen for noise bursts at the same time on a single frequency. As a result of these problems, 802.11 does not use CSMA/CD, as Ethernet does.

To deal with this problem, 802.11 supports two modes of operation. The first, called **DCF** (**Distributed Coordination Function**), does not use any kind of central control (in that respect, similar to Ethernet). The other, called **PCF** (**Point Coordination Function**), uses the base station to control all activity in its cell. All implementations must support DCF but PCF is optional. We will now discuss these two modes in turn.

The 802.11 Frame Structure

The 802.11 standard defines three different classes of frames on the wire: data, control, and management. Each of these has a header with a variety of fields used within the MAC sub-layer. In addition, there are some headers used by the physical layer but these mostly deal with the modulation techniques used, so we will not discuss them here.

First comes the Frame Control field. It itself has 11 subfields. The first of these is the Protocol version, which allows two versions of the protocol to operate at the same time in the same cell. Then come the Type (data, control, or management) and Subtype fields (e.g., RTS or CTS). The To DS and From DS bits indicate the frame is going to or coming from the intercell distribution system (e.g., Ethernet). The MF bit means that more fragments will follow. The Retry bit marks a retransmission of a frame sent earlier. The Power management bit is used by the base station to put the receiver into sleep state or take it out of sleep state. The More bit indicates that the sender has additional frames for the receiver. The W bit specifies that the frame body has been encrypted using the **WEP** (**Wired Equivalent Privacy**) algorithm. Finally, the Obit tells the receiver that a sequence of frames with this bit on must be processed strictly in order.

The second field of the data frame, the Duration field, tells how long the frame and its acknowledgement will occupy the channel. This field is also present in the control frames and is how other stations manage the NAV mechanism. The

frame header contains four addresses, all in standard IEEE 802 format. The source and destination are obviously needed, but what are the other two for? Remember that frames may enter or leave a cell via a base station. The other two addresses are used for the source and destination base stations for intercell traffic.

The Sequence field allows fragments to be numbered. Of the 16 bits available, 12 identify the frame and 4 identify the fragment. The Data field contains the payload, up to 2312 bytes, followed by the usual Checksum.

Management frames have a format similar to that of data frames, except without one of the base station addresses, because management frames are restricted to a single cell. Control frames are shorter still, having only one or two addresses, no Data field, and no Sequence field. The key information here is in the Subtype field, usually RTS, CTS, or ACK.

Services

The 802.11 standard states that each conformant wireless LAN must provide nine services. These services are divided into two categories: five distribution services and four station services. The distribution services relate to managing cell membership and interacting with stations outside the cell. In contrast, the station services relate to activity within a single cell.

The five distribution services are provided by the base stations and deal with station mobility as they enter and leave cells, attaching themselves to and detaching themselves from base stations. They are as follows.

Association:

This service is used by mobile stations to connect themselves to base stations. Typically, it is used just after a station moves within the radio range of the base station. Upon arrival, it announces its identity and capabilities. The capabilities include the data rates supported, need for PCF services (i.e., polling), and power management requirements. The base station may accept or reject the mobile station. If the mobile station is accepted, it must then authenticate itself.

Disassociation:

Either the station or the base station may disassociate, thus breakingthe relationship. A station should use this service before shutting down or leaving, but the base station may also use it before going down for maintenance.

Re-association:

A station may change its preferred base station using this service. This facility is useful for mobile stations moving from one cell to another. If it is used correctly, no data will be lost as a consequence of the handover. (But 802.11, like Ethernet, is just a best-efforts service.)

Distribution:

This service determines how to route frames sent to the base station. Ifthe destination is local to the base station, the frames can be sent out directly over the air. Otherwise, they will have to be forwarded over the wired network.

Integration:

If a frame needs to be sent through a non-802.11 network with a different addressing scheme or frame format, this service handles the translation from the 802.11 format to the format required by the destination network.

The remaining four services are intra cell (i.e., relate to actions within a single cell). They are used after association has taken place and are as follows.

Authentication:

Because wireless communication can easily be sent or received by unauthorized stations, a station must authenticate itself before it is permitted to send data. After a mobile station has been associated by the base station (i.e., accepted into its cell), the base station sends a special challenge frame to it to see if the mobile station knows the secret key (password) that has been assigned to it. It proves its knowledge of the secret key by encrypting the challenge frame and sending it back to the base station. If the result is correct, the mobile is fully enrolled in the cell. In the initial standard, the base station does not have to prove its identity to the mobile station, but work to repair this defect in the standard is underway.

De-authentication:

When a previously authenticated station wants to leave the network, it is de-authenticated. After de-authentication, it may no longer use the network.

Privacy:

For information sent over a wireless LAN to be kept confidential, it must beencrypted. This service manages the encryption and decryption. The encryption algorithm specified is RC4, invented by Ronald Rivest of M.I.T.

Data delivery:

Finally, data transmission is what it is all about, so 802.11 naturally provides a way to transmit and receive data. Since 802.11 is modeled on Ethernet and transmission over Ethernet is not guaranteed to be 100% reliable, transmission over 802.11 is not guaranteed to be reliable either. Higher layers must deal with detecting and correcting errors.

Broadband Wireless

We have been indoors too long. Let us now go outside and see if any interesting networking is going on there. It turns out that quite a bit is going on there, and some of it has to do with the so-called last mile. With the deregulation of the telephone system in many countries, competitors to the entrenched telephone company are now often allowed to offer local voice and high-speed Internet service. There is certainly plenty of demand. The problem is that running fiber, coax, or even category 5 twisted pair to millions of homes and businesses is prohibitively expensive. What is a competitor to do?

The answer is broadband wireless. Erecting a big antenna on a hill just outside of town and installing antennas directed at it on customers' roofs is much easier and cheaper than digging trenches and stringing cables. Thus, competing telecommunication companies have a great interest in providing a multimegabit wireless communication service for voice, Internet, movies on demand, etc. LMDS was invented for this purpose. However, until recently, every carrier devised its own system. This lack of standards meant that hardware and software could not be mass produced, which kept prices high and acceptance low.

Many people in the industry realized that having a broadband wireless standard was the key element missing, so IEEE was asked to form a committee composed of people from key companies and academia to draw up the standard. The next number available in the 802 numbering space was **802.16**, so the standard got this number. Work was started in July 1999, and the final standard was approved in April 2002. Officially the standard is called "Air Interface for Fixed Broadband Wireless Access Systems." However, some people prefer to call

it a **wireless MAN (Metropolitan Area Network)** or a **wireless local loop**. We regard all these terms as interchangeable.

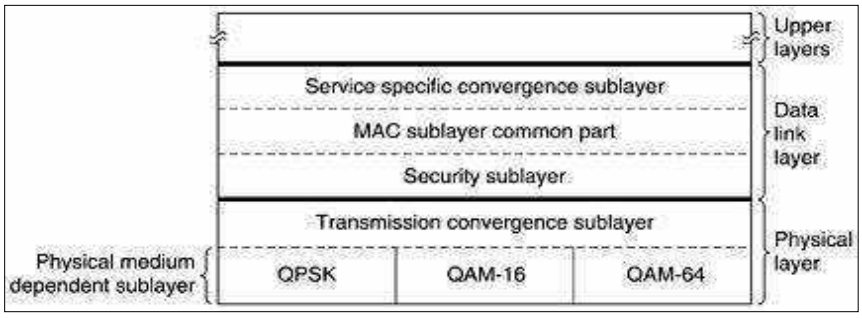
Comparison of 802.11 with 802.16

At this point you may be thinking: Why devise a new standard? Why not just use 802.11? There are some very good reasons for not using 802.11, primarily because 802.11 and 802.16 solve different problems. Before getting into the technology of 802.16, it is probably worthwhile saying a few words about why a new standard is needed at all.

The environments in which 802.11 and 802.16 operate are similar in some ways, primarily in that they were designed to provide high-bandwidth wireless communications. But they also differ in some major ways. To start with, 802.16 provides service to buildings, and buildings are not mobile. They do not migrate from cell to cell often. Much of 802.11 deals with mobility, and none of that is relevant here. Next, buildings can have more than one computer in them, a complication that does not occur when the end station is a single notebook computer. Because building owners are generally willing to spend much more money for communication gear than are notebook owners, better radios are available. This difference means that 802.16 can use full-duplex communication, something 802.11 avoids to keep the cost of the radios low.

Because 802.16 runs over part of a city, the distances involved can be several kilometers, which means that the perceived power at the base station can vary widely from station to station. This variation affects the signal-to-noise ratio, which, in, turn, dictates multiple modulation schemes. Also, open communication over a city means that security and privacy are essential and mandatory.

The 802.16 Protocol Stack



The 802.16 the general structure is similar to that of the other 802 networks, but with more sub-layers. The bottom sub-layer deals with transmission. Traditional narrow-band radio is used with conventional modulation schemes. Above the physical transmission layer comes a convergence sub-layer to hide the different technologies from the data link layer. Actually, 802.11 has something like this too, only the committee chose not to formalize it with an OSI-type name.

Although we have not shown them in the figure, work is already underway to add two new physical layer protocols. The 802.16a standard will support OFDM in the 2-to-11 GHz frequency range. The 802.16b standard will operate in the 5-GHz ISM band. Both of these are attempts to move closer to 802.11.

The data link layer consists of three sub-layers. The bottom one deals with privacy and security, which is far more crucial for public outdoor networks than for private indoor networks. It manages encryption, decryption, and key management.

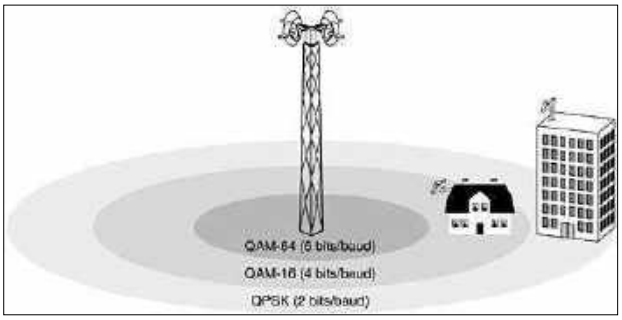
Next comes the MAC sub-layer common part. This is where the main protocols, such as channel management, are located. The model is that the base station controls the system. It can schedule the downstream (i.e., base to subscriber) channels very efficiently and plays a major role in managing the upstream (i.e., subscriber to base) channels as well. An unusual feature of the MAC sub-layer is that, unlike those of the other 802 networks, it is completely connection oriented, in order to provide quality-of-service guarantees for telephony and multimedia communication.

The service-specific convergence sub-layer takes the place of the logical link sub-layer in the other 802 protocols. Its function is to interface to the network layer. A complication here is that 802.16 was designed to integrate seamlessly with both datagram protocols (e.g., PPP, IP, and Ethernet) and ATM. The problem is that packet protocols are connectionless and ATM is connection oriented. This means that every ATM connection has to map onto an 802.16 connection, in principle a straightforward matter. But onto which 802.16 connection should an incoming IP packet be mapped? That problem is dealt with in this sub-layer.

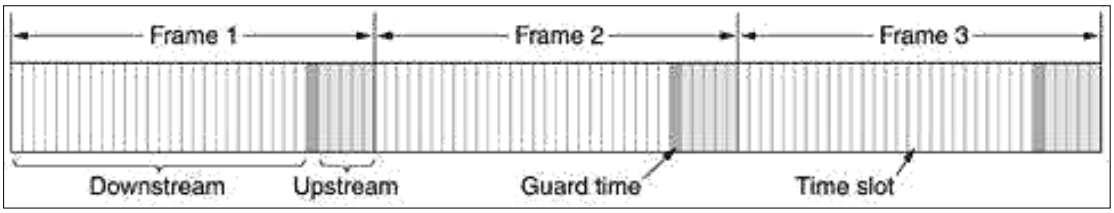
The 802.16 Physical Layer

Broadband wireless needs a lot of spectrum, and the only place to find it is in the 10-to-66 GHz range. These millimeter waves have an interesting property that longer microwaves do not: they travel in straight lines, unlike sound but similar to light. As a consequence, the base station can have multiple antennas, each pointing at a different sector of the surrounding terrain, Each sector has its own users and is fairly independent of the adjoining ones, something not true of cellular radio, which is omnidirectional.

The 802.16 transmission environment



Frames and time slots for time division duplexing



Because signal strength in the millimeter band falls off sharply with distance from the base station, the signal-to-noise ratio also drops with distance from the base station. For this reason, 802.16 employs three different modulation schemes, depending on how far the subscriber station is from the base station. For close-in subscribers, QAM-64 is used, with 6 bits/baud. For medium-distance subscribers, QAM-16 is used, with 4 bits/baud. For distant subscribers, QPSK is used, with 2 bits/baud. For example, for a typical value of 25 MHz worth of spectrum, QAM-64 gives 150 Mbps, QAM-16 gives 100 Mbps, and QPSK gives 50 Mbps. In other words, the farther the subscriber is from the base station, the lower the data rate.

Given the goal of producing a broadband system, and subject to the above physical constraints, the 802.16 designers worked hard to use the available spectrum efficiently. One thing they did not like was the way GSM and DAMPS work. Both of those use different but equal frequency bands for upstream and downstream traffic. For voice, traffic is probably symmetric for the most part, but for Internet access, there is often more downstream traffic than upstream traffic. Consequently, 802.16 provides a more flexible way to allocate the bandwidth. Two schemes are used, **FDD** (**Frequency Division Duplexing**) and **TDD** (**TimeDivision Duplexing**). Here the base station periodicallysends out frames. Each frame contains time slots. The first ones are for downstream traffic. Then comes a guard time used by the stations to switch direction. Finally, we have slots for upstream traffic. The number of time slots devoted to each direction can be changed dynamically to match the bandwidth in each direction to the traffic.

At the time a subscriber connects to a base station, they perform mutual authentication with RSA public-key cryptography using X.509 certificates. The payloads themselves are encrypted using a symmetric-key system, either DES with cipher block chaining or triple DES with two keys. AES (Rijndael) is likely to be added soon. Integrity checking uses SHA-1.

Let us now look at the MAC sub-layer common part. MAC frames occupy an integral number of physical layer time slots. Each frame is composed of sub-frames, the first two of which are the downstream and upstream maps. These maps tell what is in which time slot and which time slots are free. The downstream map also contains various system parameters to inform new stations as they come on-line.

The downstream channel is fairly straightforward. The base station simply decides what to put in which sub-frame. The upstream channel is more complicated since there are competing uncoordinated subscribers that need access to it. Its allocation is tied closely to the quality-of-service issue.

Four classes of service are defined as follows:

- Constant bit rate service.
- Real-time variable bit rate service.
- Non-real-time variable bit rate service.
- Best-efforts service.

All service in 802.16 is connection-oriented, and each connection gets one of the above classes of service, determined when the connection is set up. This design is very different from that of 802.11 or Ethernet, which have no connections in the MAC sub-layer.

Constant bit rate service is intended for transmitting uncompressed voice such as on a T1 channel. This service needs to send a predetermined amount of data at predetermined time intervals. It is accommodated by dedicating certain time slots to each connection of this type. Once the bandwidth has been allocated, the time slots are available automatically, without the need to ask for each one.

Real-time variable bit rate service is for compressed multimedia and other soft real-time applications in which the amount of bandwidth needed each instant may vary. It is accommodated by the base station polling the subscriber at a fixed interval to ask how much bandwidth is needed this time.

Non-real-time variable bit rate service is for heavy transmissions that are not real time, such as large file transfers. For this service the base station polls the subscriber often, but not at rigidly-prescribed time intervals. A constant bit rate customer can set a bit in one of its frames requesting a poll in order to send additional (variable bit rate) traffic.

If a station does not respond to a poll k times in a row, the base station puts it into a multicast group and takes away its personal poll. Instead, when the multicast group is polled, any of the stations in it can respond, contending for service. In this way, stations with little traffic do not waste valuable polls.

Finally, best-efforts service is for everything else. No polling is done and the subscriber must contend for bandwidth with other best-efforts subscribers. Requests for bandwidth are done in time slots marked in the upstream map as available for contention. If a request is successful, its success will be noted in the next downstream map. If it is not successful, unsuccessful subscribers have to try again later. To minimize collisions, the Ethernet binary exponential back-off algorithm is used.

The standard defines two forms of bandwidth allocation: per station and per connection. In the former case, the subscriber station aggregates the needs of all the users in the building and makes collective requests for them. When it is

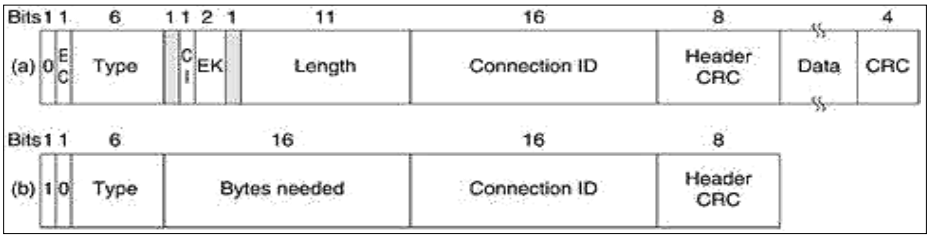
granted bandwidth, it doles out that bandwidth to its users as it sees fit. In the latter case, the base station manages each connection directly.

The 802.16 Frame Structure

All MAC frames begin with a generic header. The header is followed by an optional payload and an optional checksum (CRC). The payload is not needed in control frames, for example, those requesting channel slots. The checksum is (surprisingly) also optional due to the error correction in the physical layer and the fact that no attempt is ever made to retransmit real-time frames. If no retransmissions will be attempted, why even bother with a checksum?

A quick rundown of the header fields of the EC bit tells whether the payload is encrypted. The Type field identifies the frame type, mostly telling whether packing and fragmentation are present. The CI field indicates the presence or absence of the final checksum. The EK field tells which of the encryption keys is being used (if any). The Length field gives the complete length of the frame, including the header. The Connection identifier tells which connection this frame belongs to. Finally, the Header CRC field is a checksum over the header only, using the polynomial $x^8 + x^2 + x + 1$.

(a) A generic frame.
(b) A bandwidth request frame



A second header type, for frames that request bandwidth, It starts with a 1 bit instead of a 0 bit and is similar to the generic header except that the second and third bytes form a 16-bit number telling how much bandwidth is needed to carry the specified number of bytes. Bandwidth request frames do not carry a payload or full-frame CRC.

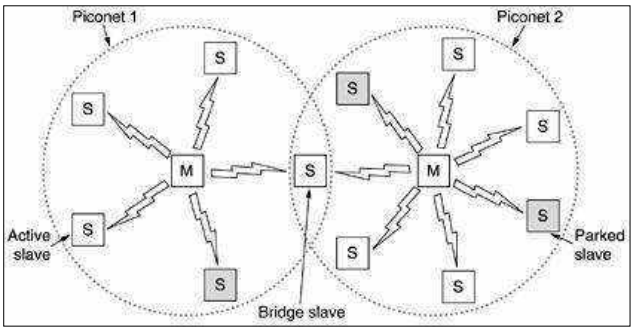
A great deal more could be said about 802.16, but this is not the place to say it. For more information, please consult the standard itself.

Two piconets can be connected to form a scatternet.

Bluetooth

Bluetooth Architecture

Let us start our study of the Bluetooth system with a quick overview of what it contains and what it is intended to do. The basic unit of a Bluetooth system is a piconet, which consists of a master node and up to seven active slave nodes within a distance of 10 meters. Multiple piconets can exist in the same (large) room and can even be connected via a bridge node, as shown in Fig. An interconnected collection of piconets is called a scatternet Two piconets can be connected to form a scatter net



In addition to the seven active slave nodes in a piconet, there can be up to 255 parked nodes in the net. These are devices that the master has switched to a low-power state to reduce the drain on their batteries. In parked state, a device cannot do anything except respond to an activation or beacon signal from the master. There are also two intermediate power states, hold and sniff, but these will not concern us here.

The Bluetooth profiles

Name	Description
Generic access	Procedures for link management
Service discovery	Protocol for discovering offered services
Serial port	Replacement for a serial port cable
Generic object exchange	Defines client-server relationship for object movement
LAN access	Protocol between a mobile computer and a fixed LAN
Dial-up networking	Allows a notebook computer to call via a mobile phone
Fax	Allows a mobile fax machine to talk to a mobile phone
Cordless telephony	Connects a handset and its local base station
Intercom	Digital walkie-talkie
Headset	Allows hands-free voice communication
Object push	Provides a way to exchange simple objects
File transfer	Provides a more general file transfer facility
Synchronization	Permits a PDA to synchronize with another computer

The reason for the master/slave design is that the designers intended to facilitate the implementation of complete Bluetooth chips for under $5. The consequence of this decision is that the slaves are fairly dumb, basically just doing whatever the master tells them to do. At its heart, a piconet is a centralized TDM system, with the master controlling the clock and determining which device gets to communicate in which time slot. All communication is between the master and a slave; direct slave-slave communication is not possible.

Bluetooth Applications

Most network protocols just provide channels between communicating entities and let applications designers figure out what they want to use them for. For example, 802.11 does not specify whether users should use their notebook computers for reading e-mail, surfing the Web, or something else. In contrast, the Bluetooth V1.1 specification names 13 specific applications to be supported and provides different protocol stacks for each one. Unfortunately, this approach leads to a very large amount of complexity, which we will omit here. The 13 applications, which are called **profiles**, By looking at them briefly now, we may see more clearly what the Bluetooth SIG is trying to accomplish.

The generic access profile is not really an application, but rather the basis upon which the real applications are built. Its main job is to provide a way to establish and maintain secure links (channels) between the master and the slaves. Also relatively generic is the service discovery profile, which is used by devices to discover what services other devices have to offer. All Bluetooth devices are expected to implement these two profiles. The remaining ones are optional.

The serial port profile is a transport protocol that most of the remaining profiles use. It emulates a serial line and is especially useful for legacy applications that expect a serial line.

The generic object exchange profile defines a client-server relationship for moving data around. Clients initiate operations, but a slave can be either a client or a server. Like the serial port profile, it is a building block for other profiles.

The next group of three profiles is for networking. The LAN access profile allows a Bluetooth device to connect to a fixed network. This profile is a direct competitor to 802.11. The dial-up networking profile was the original motivation for the whole project. It allows a notebook computer to connect to a mobile phone

containing a built-in modem without wires. The fax profile is similar to dial-up networking, except that it allows wireless fax machines to send and receive faxes using mobile phones without a wire between the two.

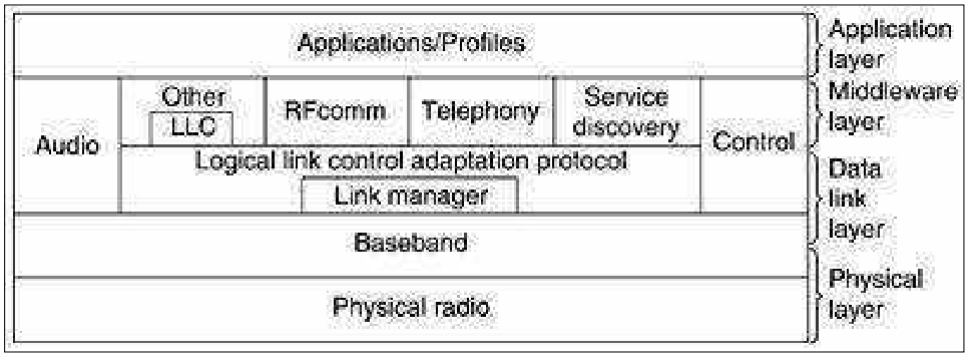
The next three profiles are for telephony. The cordless telephony profile provides a way to connect the handset of a cordless telephone to the base station. Currently, most cordless telephones cannot also be used as mobile phones, but in the future, cordless and mobile phones may merge. The intercom profile allows two telephones to connect as walkie-talkies. Finally, the headset profile provides hands-free voice communication between the headset and its base station, for example, for hands-free telephony while driving a car.

The remaining three profiles are for actually exchanging objects between two wireless devices. These could be business cards, pictures, or data files. The synchronization profile, in particular, is intended for loading data into a PDA or notebook computer when it leaves home and collecting data from it when it returns.

The Bluetooth Protocol Stack

The Bluetooth standard has many protocols grouped loosely into layers. The layer structure does not follow the OSI model, the TCP/IP model, the 802 model, or any other known model. However, IEEE is working on modifying Bluetooth to shoehorn it into the 802 model better. The basic Bluetooth protocol architecture as modified by the 802 committee is shown in Fig.

The 802.15 version of the Bluetooth protocol architecture



The bottom layer is the physical radio layer, which corresponds fairly well to the physical layer in the OSI and 802 models. It deals with radio transmission and modulation. Many of the concerns here have to do with the goal of making

the system inexpensive so that it can become a mass market item.The baseband layer is somewhat analogous to the MAC sub-layer but also includes elements of the physical layer. It deals with how the master controls time slots and how these slots are grouped into frames.

Next comes a layer with a group of somewhat related protocols. The link manager handles the establishment of logical channels between devices, including power management, authentication, and quality of service. The logical link control adaptation protocol (often called L2CAP) shields the upper layers from the details of transmission. It is analogous to the standard 802 LLC sub-layer, but technically different from it. As the names suggest, the audio and control protocols deal with audio and control, respectively. The applications can get at them directly, without having to go through the L2CAP protocol.

The next layer up is the middleware layer, which contains a mix of different protocols. The 802 LLC was inserted here by IEEE for compatibility with its other 802 networks. The RFcomm, telephony, and service discovery protocols are native. RFcomm (Radio Frequency communication) is the protocol that emulates the standard serial port found on PCs for connecting the keyboard, mouse, and modem, among other devices. It has been designed to allow legacy devices to use it easily. The telephony protocol is a real-time protocol used for the three speech-oriented profiles. It also manages call setup and termination. Finally, the service discovery protocol is used to locate services within the network.

The top layer is where the applications and profiles are located. They make use of the protocols in lower layers to get their work done. Each application has its own dedicated subset of the protocols. Specific devices, such as a headset, usually contain only those protocols needed by that application and no others.

The Bluetooth Radio Layer

The radio layer moves the bits from master to slave, or vice versa. It is a low-power system with a range of 10 meters operating in the 2.4-GHz ISM band. The band is divided into 79 channels of 1 MHz each. Modulation is frequency shift keying, with 1 bit per Hz giving a gross data rate of 1 Mbps, but much of this spectrum is consumed by overhead. To allocate the channels fairly, frequency hopping spread spectrum is used with 1600 hops/sec and a dwell time of 625

μsec. All the nodes in a piconet hop simultaneously, with the master dictating the hop sequence.

Because both 802.11 and Bluetooth operate in the 2.4-GHz ISM band on the same 79 channels, they interfere with each other. Since Bluetooth hops far faster than 802.11, it is far more likely that a Bluetooth device will ruin 802.11 transmissions than the other way around. Since 802.11 and 802.15 are both IEEE standards, IEEE is looking for a solution to this problem, but it is not so easy to find since both systems use the ISM band for the same reason: no license is required there. The 802.11a standard uses the other (5 GHz) ISM band, but it has a much shorter range than 802.11b (due to the physics of radio waves), so using 802.11a is not a perfect solution for all cases. Some companies have solved the problem by banning Bluetooth altogether. A market-based solution is for the network with more power (politically and economically, not electrically) to demand that the weaker party modify its standard to stop interfering with it. Some thoughts on this matter are given in (Lansford et al., 2001).

The Bluetooth Baseband Layer

The baseband layer is the closest thing Bluetooth has to a MAC sub-layer. It turns the raw bit stream into frames and defines some key formats. In the simplest form, the master in each piconet defines a series of 625 μsec time slots, with the master's transmissions starting in the even slots and the slaves' transmissions starting in the odd ones. This is traditional time division multiplexing, with the master getting half the slots and the slaves sharing the other half. Frames can be 1, 3, or 5 slots long.

The frequency hopping timing allows a settling time of 250–260 μsec per hop to allow the radio circuits to become stable. Faster settling is possible, but only at higher cost. For a single-slot frame, after settling, 366 of the 625 bits are left over. Of these, 126 are for an access code and the header, leaving 240 bits for data. When five slots are strung together, only one settling period is needed and a slightly shorter settling period is used, so of the 5 x 625 =3125 bits in five time slots, 2781 are available to the baseband layer. Thus, longer frames are much more efficient than single-slot frames.

Each frame is transmitted over a logical channel, called a **link**, between the master and a slave. Two kinds of links exist. The first is the **ACL** (**Asynchronous Connection-Less**) link, which is used for packet-switched data available at

irregular intervals. These data come from the L2CAP layer on the sending side and are delivered to the L2CAP layer on the receiving side. ACL traffic is delivered on a best-efforts basis. No guarantees are given. Frames can be lost and may have to be retransmitted. A slave may have only one ACL link to its master.

The other is the **SCO** (**Synchronous Connection Oriented**) link, for real-time data, such as telephone connections. This type of channel is allocated a fixed slot in each direction. Due to the time-critical nature of SCO links, frames sent over them are never retransmitted. Instead, forward error correction can be used to provide high reliability. A slave may have up to three SCO links with its master. Each SCO link can transmit one 64,000 bps PCM audio channel.

The Bluetooth L2CAP Layer

The L2CAP layer has three major functions. First, it accepts packets of up to 64 KB from the upper layers and breaks them into frames for transmission. At the far end, the frames are reassembled into packets again.

Second, it handles the multiplexing and de-multiplexing of multiple packet sources. When a packet has been reassembled, the L2CAP layer determines which upper-layer protocol to hand it to, for example, RFcomm or telephony.

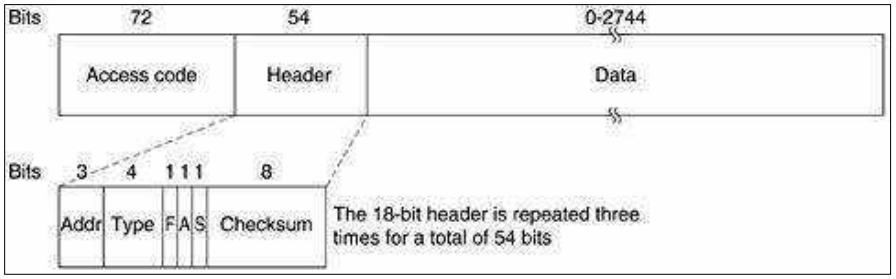
Third, L2CAP handles the quality of service requirements, both when links are established and during normal operation. Also negotiated at setup time is the maximum payload size allowed, to prevent a large-packet device from drowning a small-packet device. This feature is needed because not all devices can handle the 64-KB maximum packet.

The Bluetooth Frame Structure

A typical Bluetooth data frame.

There are several frame formats. It begins with an access code that usually identifies the master so that slaves within radio range of two masters can tell which traffic is for them. Next comes a 54-bit header containing typical MAC sub-layer fields. Then comes the data field, of up to 2744 bits (for a five-slot transmission). For a single time slot, the format is the same except that the data field is 240 bits.

A typical Bluetooth data frame



Let us take a quick look at the header. The Address field identifies which of the eight active devices the frame is intended for. The Type field identifies the frame type (ACL, SCO, poll, or null), the type of error correction used in the data field, and how many slots long the frame is. The Flow bit is asserted by a slave when its buffer is full and cannot receive any more data. This is a primitive form of flow control. The Acknowledgement bit is used to piggyback an ACK onto a frame. The Sequence bit is used to number the frames to detect retransmissions. The protocol is stop-and-wait, so 1 bit is enough. Then comes the 8-bit header Checksum. The entire 18-bit header is repeated three times to form the 54-bit header. On the receiving side, a simple circuit examines all three copies of each bit. If all three are the same, the bit is accepted. If not, the majority opinion wins. Thus, 54 bits of transmission capacity are used to send 10 bits of header. The reason is that to reliably send data in a noisy environment using cheap, low-powered (2.5 mW) devices with little computing capacity, a great deal of redundancy is needed.

Various formats are used for the data field for ACL frames. The SCO frames are simpler though: the data field is always 240 bits. Three variants are defined, permitting 80, 160, or 240 bits of actual payload, with the rest being used for error correction. In the most reliable version (80-bit payload), the contents are just repeated three times, the same as the header.

Since the slave may use only the odd slots, it gets 800 slots/sec, just as the master does. With an 80-bit payload, the channel capacity from the slave is 64,000 bps and the channel capacity from the master is also 64,000 bps, exactly enough for a single full-duplex PCM voice channel (which is why a hop rate of 1600 hops/sec was chosen). These numbers mean that a full-duplex voice channel with 64,000 bps in each direction using the most reliable format completely saturates the piconet despite a raw bandwidth of 1 Mbps. For the least reliable variant (240

bits/slot with no redundancy at this level), three full-duplex voice channels can be supported at once, which is why a maximum of three SCO links is permitted per slave.

There is much more to be said about Bluetooth, but no more space to say it here. For more information, see (Bhagwat, 2001; Bisdikian, 2001; Bray and Sturman, 2002; Haartsen, 2000; Johansson et al., 2001; Miller and Bisdikian, 2001; and Sairam et al., 2002).

UNIT - IV
NETWORK & TRANSPORT LAYER

Routing Algorithms

The main function of the network layer is routing packets from the source machine to the destination machine. In most subnets, packets will require multiple hops to make the journey. The only notable exception is for broadcast networks, but even here routing is an issue if the source and destination are not on the same network. The algorithms that choose the routes and the data structures that they use are a major area of network layer design.

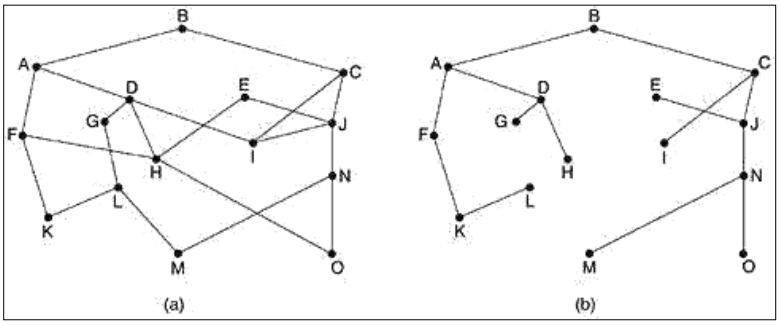
The routing algorithm is that part of the network layer software responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagrams internally, this decision must be made anew for every arriving data packet since the best route may have changed since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

The routing algorithm is that part of the network layer software responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagrams internally, this decision must be made anew for every arriving data packet since the inks. To maximize the total flow, the X to X' traffic should be shut off altogether. Unfortunately, X and X' may not see it that way. Evidently, some compromise between global efficiency and fairness to individual connections is needed.

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

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

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



5.2.1 The Optimality Principle

Before we get into specific algorithms, it may be helpful to note that one can make a general statement about optimal routes without regard to network topology or traffic. This statement is known as the **optimality principle**. It states that if router J is on the optimal path from router I to router K, then the optimal path from J to K also falls along the same route. To see this,call the part of the route from I to J r_1 and the rest of the route r_2. If a route better than r_2 existed from J to K, it could be concatenated with r_1 to improve the route from I to K, contradicting our statement that r_1r_2 is optimal.

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

Shortest Path Routing

Let us begin our study of feasible routing algorithms with a technique that is widely used in many forms because it is simple and easy to understand. The idea is to build a graph of the subnet, with each node of the graph representing a

router and each arc of the graph representing a communication line (often called a link). To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.

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

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

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

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

We want to find the shortest path from A to D. We start out by marking node A as permanent, indicated by a filled-in circle. Then we examine, in turn, each of the nodes adjacent to A (the working node), relabeling each one with the distance to A. Whenever a node is relabeled, we also label it with the node from which the probe was made so that we can reconstruct the final path later. Having examined each of the nodes adjacent to A, we examine all the tentatively labeled

nodes in the whole graph and make the one with the smallest label permanent, This one becomes the new working node.

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

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

Suppose that there were a shorter path than ABE, say AXYZE. There are two possibilities: either node Z has already been made permanent, or it has not been. If it has, then E has already been probed (on the round following the one when Z was made permanent), so the AXYZE path has not escaped our attention and thus cannot be a shorter path.

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

Distance Vector Routing

Modern computer networks generally use dynamic routing algorithms rather than the static ones described above because static algorithms do not take the current network load into account. Two dynamic algorithms in particular, distance vector routing and link state routing, are the most popular. In this section we will look at the former algorithm. In the following section we will study the latter algorithm.

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

The distance vector routing algorithm is sometimes called by other names, most commonly the distributed Bellman-Ford routing algorithm and the Ford-

Fulkerson algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the name RIP.

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

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

Link State Routing

Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing. Two primary problems caused its demise. First, since the delay metric was queue length, it did not take line bandwidth into account when choosing routes. Initially, all the lines were 56 kbps, so line bandwidth was not an issue, but after some lines had been upgraded to 230 kbps and others to 1.544 Mbps, not taking bandwidth into account was a major problem. Of course, it would have been possible to change the delay metric to factor in line bandwidth, but a second problem also existed, namely, the algorithm often took too long to converge (the count-to-infinity problem). For these reasons, it was replaced by an entirely new algorithm, now called **link state routing**. Variants of link state routing are now widely used.

The idea behind link state routing is simple and can be stated as five parts. Each router must do the following:

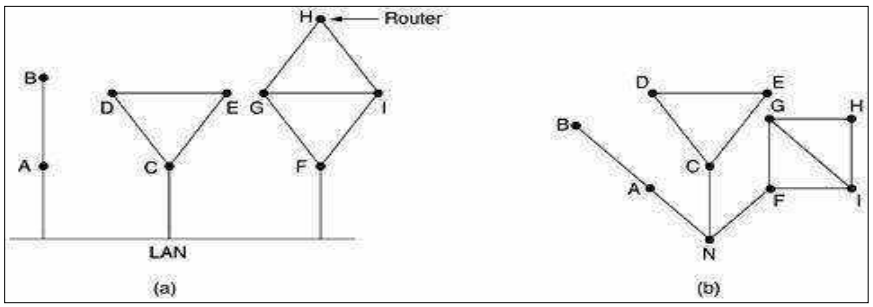
- Discover its neighbors and learn their network addresses.
- Measure the delay or cost to each of its neighbors.
- Construct a packet telling all it has just learned.
- Send this packet to all other routers.
- Compute the shortest path to every other router.

In effect, the complete topology and all delays are experimentally measured and distributed to every router. Then Dijkstra's algorithm can be run to find the shortest path to every other router. Below we will consider each of these five steps in more detail.

Learning about the Neighbors

When a router is booted, its first task is to learn who its neighbors are. It accomplishes this goal by sending a special HELLO packet on each point-to-point line. The router on the other end is expected to send back a reply telling who it is. These names must be globally unique because when a distant router later hears that three routers are all connected to F, it is essential that it can determine whether all three mean the same F.

(a) Nine routers and a LAN. (b) A graph model of (a)



When two or more routers are connected by a LAN, the situation is slightly more complicated. A LAN to which three routers,A,C, andF, are directly connected. Eachof these routers is connected to one or more additional routers, as shown.

One way to model the LAN is to consider it as a node itself, Here we have introduced a new, artificial node, N, to which A, C, and F are connected. The fact that it is possible to go from A to C on the LAN is represented by the path ANC here.

Measuring Line Cost

The link state routing algorithm requires each router to know, or at least have a reasonable estimate of, the delay to each of its neighbors. The most direct way to determine this delay is to send over the line a special ECHO packet that the other side is required to send back immediately. By measuring the round-trip time and dividing it by two, the sending router can get a reasonable estimate of

the delay. For even better results, the test can be conducted several times, and the average used. Of course, this method implicitly assumes the delays are symmetric, which may not always be the case.

An interesting issue is whether to take the load into account when measuring the delay. To factor the load in, the round-trip timer must be started when the ECHO packet is queued. To ignore the load, the timer should be started when the ECHO packet reaches the front of the queue.

Building Link State Packets

Once the information needed for the exchange has been collected, the next step is for each router to build a packet containing all the data. The packet starts with the identity of the sender, followed by a sequence number and age (to be described later), and a list of neighbors. For each neighbor, the delay to that neighbor is given. The corresponding link state packets for all six routers.

Building the link state packets is easy. The hard part is determining when to build them. One possibility is to build them periodically, that is, at regular intervals. Another possibility is to build them when some significant event occurs, such as a line or neighbor going down or coming back up again or changing its properties appreciably.

Distributing the Link State Packets

The trickiest part of the algorithm is distributing the link state packets reliably. As the packets are distributed and installed, the routers getting the first ones will change their routes. Consequently, the different routers may be using different versions of the topology, which can lead to inconsistencies, loops, unreachable machines, and other problems.

First we will describe the basic distribution algorithm. Later we will give some refinements. The fundamental idea is to use flooding to distribute the link state packets. To keep the flood in check, each packet contains a sequence number that is incremented for each new packet sent. Routers keep track of all the (source router, sequence) pairs they see. When a new link state packet comes in, it is checked against the list of packets already seen. If it is new, it is forwarded on all lines except the one it arrived on. If it is a duplicate, it is discarded. If a packet with a sequence number lower than the highest one seen so far ever arrives, it is rejected as being obsolete since the router has more recent data.

This algorithm has a few problems, but they are manageable. First, if the sequence numbers wrap around, confusion will reign. The solution here is to use a 32-bit sequence number. With one link state packet per second, it would take 137 years to wrap around, so this possibility can be ignored.

Second, if a router ever crashes, it will lose track of its sequence number. If it starts again at 0, the next packet will be rejected as a duplicate.

Third, if a sequence number is ever corrupted and 65,540 is received instead of 4 (a 1-bit error), packets 5 through 65,540 will be rejected as obsolete, since the current sequence number is thought to be 65,540.

Computing the New Routes

Once a router has accumulated a full set of link state packets, it can construct the entire subnet graph because every link is represented. Every link is, in fact, represented twice, once for each direction. The two values can be averaged or used separately.

Now Dijkstra's algorithm can be run locally to construct the shortest path to all possible destinations. The results of this algorithm can be installed in the routing tables, and normal operation resumed.

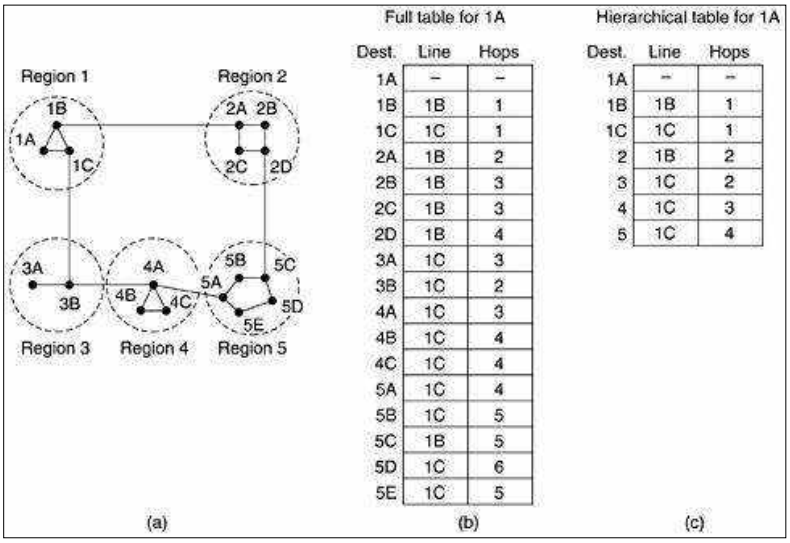
For a subnet with n routers, each of which has k neighbors, the memory required to store the input data is proportional to kn. For large subnets, this can be a problem. Also, the computation time can be an issue. Nevertheless, in many practical situations, link state routing works well.

Another link state protocol is **IS-IS** (**Intermediate System-Intermediate System**), which was designed for DECnet and later adopted by ISO for use with its connectionless network layer protocol, CLNP. Since then it has been modified to handle other protocols as well, most notably, IP. IS-IS is used in some Internet backbones (including the old NSFNET backbone) and in some digital cellular systems such as CDPD. Novell NetWare uses a minor variant of IS-IS (NLSP) for routing IPX packets.

Hierarchical Routing

As networks grow in size, the router routing tables grow proportionally. Not only is router memory consumed by ever-increasing tables, but more CPU time is needed to scan them and more bandwidth is needed to send status reports about them. At a certain point the network may grow to the point where it is no longer feasible for every router to have an entry for every other router, so the routing will have to be done hierarchically, as it is in the telephone network.

Hierarchical routing



Full table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

Hierarchical table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

When hierarchical routing is used, the routers are divided into what we will call regions, with each router knowing all the details about how to route packets to destinations within its own region, but knowing nothing about the internal structure of other regions. When different networks are interconnected, it is natural to regard each one as a separate region in order to free the routers in one network from having to know the topological structure of the other ones.

Broadcast Routing

In some applications, hosts need to send messages to many or all other hosts. For example, a service distributing weather reports, stock market updates, or live radio programs might work best by broadcasting to all machines and letting those that are interested read the data.

Sending a packet to all destinations simultaneously is called **broadcasting**; various methods have been proposed for doing it. One broadcasting method that

requires no special features from the subnet is for the source to simply send a distinct packet to each destination. Not only is the method wasteful of bandwidth, but it also requires the source to have a complete list of all destinations. In practice this may be the only possibility, but it is the least desirable of the methods.

Flooding is another obvious candidate. Although flooding is ill-suited for ordinary point-to-point communication, for broadcasting it might rate serious consideration, especially if none of the methods described below are applicable. The problem with flooding as a broadcast technique is the same problem it has as a point-to-point routing algorithm: it generates too many packets and consumes too much bandwidth.

A third algorithm is multi-destination routing. If this method is used, each packet contains either a list of destinations or a bit map indicating the desired destinations. When a packet arrives at a router, the router checks all the destinations to determine the set of output lines that will be needed. (An output line is needed if it is the best route to at least one of the destinations.) The router generates a new copy of the packet for each output line to be used and includes in each packet only those destinations that are to use the line. In effect, the destination set is partitioned among the output lines. After a sufficient number of hops, each packet will carry only one destination and can be treated as a normal packet. Multi-destination routing is like separately addressed packets, except that when several packets must follow the same route, one of them pays full fare and the rest ride free.

A fourth broadcast algorithm makes explicit use of the sink tree for the router initiating the broadcast—or any other convenient spanning tree for that matter. A spanning tree is a subset of the subnet that includes all the routers but contains no loops. If each router knows which of its lines belong to the spanning tree, it can copy an incoming broadcast packet onto all the spanning tree lines except the one it arrived on. This method makes excellent use of bandwidth, generating the absolute minimum number of packets necessary to do the job. The only problem is that each router must have knowledge of some spanning tree for the method to be applicable. Sometimes this information is available (e.g., with link state routing) but sometimes it is not (e.g., with distance vector routing).

Our last broadcast algorithm is an attempt to approximate the behavior of the previous one, even when the routers do not know anything at all about

spanning trees. The idea, called reverse path forwarding, is remarkably simple once it has been pointed out. When a broadcast packet arrives at a router, the router checks to see if the packet arrived on the line that is normally used for sending packets to the source of the broadcast. If so, there is an excellent chance that the broadcast packet itself followed the best route from the router and is therefore the first copy to arrive at the router. This being the case, the router forwards copies of it onto all lines except the one it arrived on. If, however, the broadcast packet arrived on a line other than the preferred one for reaching the source, the packet is discarded as a likely duplicate.

Multicast Routing

Some applications require that widely-separated processes work together in groups, for example, a group of processes implementing a distributed database system. In these situations, it is frequently necessary for one process to send a message to all the other members of the group. If the group is small, it can just send each other member a point-to-point message. If the group is large, this strategy is expensive. Sometimes broadcasting can be used, but using broadcasting to inform 1000 machines on a million-node network is inefficient because most receivers are not interested in the message (or worse yet, they are definitely interested but are not supposed to see it). Thus, we need a way to send messages to well-defined groups that are numerically large in size but small compared to the network as a whole.

Sending a message to such a group is called multicasting, and its routing algorithm is called multicast routing. In this section we will describe one way of doing multicast routing. For additional information, see (Chu et al., 2000; Costa et al. 2001; Kasera et al., 2000; Madruga and Garcia-Luna-Aceves, 2001; Zhang and Ryu, 2001).

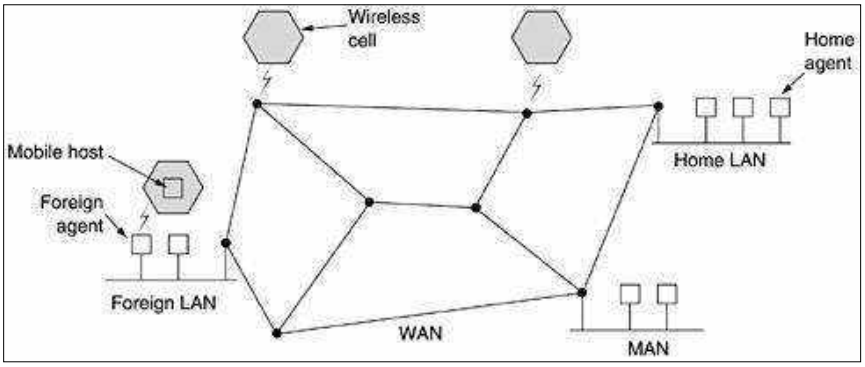
Multicasting requires group management. Some way is needed to create and destroy groups, and to allow processes to join and leave groups. How these tasks are accomplished is not of concern to the routing algorithm. What is of concern is that when a process joins a group, it informs its host of this fact. It is important that routers know which of their hosts belong to which groups. Either hosts must inform their routers about changes in group membership, or routers must query their hosts periodically. Either way, routers learn about which of their hosts are in which groups. Routers tell their neighbors, so the information propagates through the subnet.

To do multicast routing, each router computes a spanning tree covering all other routers. We have two groups, 1 and 2. Some routers are attached to hosts that belong to one or both of these groups, as indicated in the figure. A spanning tree for the leftmost router.

5.2.9 Routing for Mobile Hosts

Millions of people have portable computers nowadays, and they generally want to read their e-mail and access their normal file systems wherever in the world they may be. These mobilehosts introduce a new complication: to route a packet to a mobile host, the network first has to find it. The subject of incorporating mobile hosts into a network is very young, but in this section we will sketch some of the issues and give a possible solution.

A WAN to which LANs, MANs, and wireless cells are attached



The model of the world that network designers typically use. Here we have a WAN consisting of routers and hosts. Connected to the WAN are LANs, MANs, and wireless cells.

Hosts that never move are said to be stationary. They are connected to the network by copper wires or fiber optics. In contrast, we can distinguish two other kinds of hosts. Migratory hosts are basically stationary hosts who move from one fixed site to another from time to time but use the network only when they are physically connected to it. Roaming hosts actually compute on the run and want to maintain their connections as they move around. We will use the term **mobile hosts** to mean either of the latter two categories, that is, all hosts that are away from home and still want to be connected.

Routing in Ad Hoc Networks

We have now seen how to do routing when the hosts are mobile but the routers are fixed. An even more extreme case is one in which the routers themselves are mobile. Among the possibilities are:

- Military vehicles on a battlefield with no existing infrastructure.
- A fleet of ships at sea.
- Emergency workers at an earthquake that destroyed the infrastructure.
- A gathering of people with notebook computers in an area lacking 802.11.

In all these cases, and others, each node consists of a router and a host, usually on the same computer. Networks of nodes that just happen to be near each other are called **ad hocnet works** or **MANETs** (**Mobile Ad hoc NETworks**). Let us now examine them briefly. More information can be found in (Perkins, 2001).

A variety of routing algorithms for ad hoc networks have been proposed. One of the more interesting ones is the **AODV** (**Ad hoc On-demand Distance Vector**) routing algorithm (Perkins and Royer, 1999). It is a distant relative of the Bellman-Ford distance vector algorithm but adapted to work in a mobile environment and takes into account the limited bandwidth and low battery life found in this environment. Another unusual characteristic is that it is an on-demand algorithm, that is, it determines a route to some destination only when somebody wants to send a packet to that destination.

Route Discovery

At any instant of time, an ad hoc network can be described by a graph of the nodes (routers + hosts). Two nodes are connected (i.e., have an arc between them in the graph) if they can communicate directly using their radios. Since one of the two may have a more powerful transmitter than the other, it is possible that A is connected to B but B is not connected to A. However, for simplicity, we will assume all connections are symmetric. It should also be noted that the mere fact that two nodes are within radio range of each other does not mean that they are connected. There may be buildings, hills, or other obstacles that block their communication.

Route Maintenance

Because nodes can move or be switched off, the topology can change spontaneously. For example, if G is switched off, A will not realize that the route it was using to I (ADGI) is no longer valid. The algorithm needs to be able to deal with this. Periodically, each node broadcasts a Hello message. Each of its neighbors is expected to respond to it. If no response is forthcoming, the broadcaster knows that that neighbor has moved out of range and is no longer connected to it. Similarly, if it tries to send a packet to a neighbor that does not respond, it learns that the neighbor is no longer available.

This information is used to purge routes that no longer work. For each possible destination, each node, N, keeps track of its neighbors that have fed it a packet for that destination during the last ΔT seconds. These are called N's **active neighbors** for that destination. N does this by having a routing table keyed by destination and containing the outgoing node to use to reach the destination, the hop count to the destination, the most recent destination sequence number, and the list of active neighbors for that destination.

Congestion Control Algorithms

General Principles of Congestion Control

In contrast, closed loop solutions are based on the concept of a feedback loop. This approach has three parts when applied to congestion control:

- Monitor the system to detect when and where congestion occurs.
- Pass this information to places where action can be taken.
- Adjust system operation to correct the problem.

A variety of metrics can be used to monitor the subnet for congestion. Chief among these are the percentage of all packets discarded for lack of buffer space, the average queue lengths, the number of packets that time out and are retransmitted, the average packet delay, and the standard deviation of packet delay. In all cases, rising numbers indicate growing congestion.

The second step in the feedback loop is to transfer the information about the congestion from the point where it is detected to the point where something can be done about it. The obvious way is for the router detecting the congestion to send a packet to the traffic source or sources, announcing the problem. Of course, these extra packets increase the load at precisely the moment that more load is not needed, namely, when the subnet is congested.

Congestion Prevention Policies

Let us begin our study of methods to control congestion by looking at open loop systems. These systems are designed to minimize congestion in the first place, rather than letting it happen and reacting after the fact. They try to achieve their goal by using appropriate policies at various levels.

Policies that affect congestion

Layer	Policies
Transport	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy • Timeout determination
Network	• Virtual circuits versus datagram inside the subnet • Packet queueing and service policy • Packet discard policy • Routing algorithm • Packet lifetime management
Data link	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy

Let us start at the data link layer and work our way upward. The retransmission policy is concerned with how fast a sender times out and what it transmits upon timeout. A jumpy sender that times out quickly and retransmits all outstanding packets using go back n will put a heavier load on the system than will a leisurely sender that uses selective repeat. Closely related to this is the buffering policy. If receivers routinely discard all out-of-order packets, these packets will have to be transmitted again later, creating extra load. With respect to congestion control, selective repeat is clearly better than go back.

Acknowledgement policy also affects congestion. If each packet is acknowledged immediately, the acknowledgement packets generate extra traffic. However, if acknowledgements are saved up to piggyback onto reverse traffic, extra timeouts and retransmissions may result. A tight flow control scheme (e.g., a small window) reduces the data rate and thus helps fight congestion.

At the network layer, the choice between using virtual circuits and using datagrams affects congestion since many congestion control algorithms work only with virtual-circuit subnets. Packet queueing and service policy relates to whether routers have one queue per input line, one queue per output line, or both.

It also relates to the order in which packets are processed (e.g., round robin or priority based). Discard policy is the rule telling which packet is dropped when there is no space. A good policy can help alleviate congestion and a bad one can make it worse.

A good routing algorithm can help avoid congestion by spreading the traffic over all the lines, whereas a bad one can send too much traffic over already congested lines. Finally, packet lifetime management deals with how long a packet may live before being discarded. If it is too long, lost packets may clog up the works for a long time, but if it is too short, packets may sometimes time out before reaching their destination, thus inducing retransmissions.

In the transport layer, the same issues occur as in the data link layer, but in addition, determining the timeout interval is harder because the transit time across the network is less predictable than the transit time over a wire between two routers. If the timeout interval is too short, extra packets will be sent unnecessarily. If it is too long, congestion will be reduced but the response time will suffer whenever a packet is lost.

Congestion Control in Virtual-Circuit Subnets

The congestion control methods described above are basically open loop: they try to prevent congestion from occurring in the first place, rather than dealing with it after the fact. In this section we will describe some approaches to dynamically controlling congestion in virtual-circuit subnets. In the next two, we will look at techniques that can be used in any subnet.

One technique that is widely used to keep congestion that has already started from getting worse is **admission control**. The idea is simple: once congestion has been signaled, no more virtual circuits are set up until the problem has gone away. Thus, attempts to set up new transport layer connections fail. Letting more people in just makes matters worse. While this approach is crude, it is simple and easy to carry out. In the telephone system, when a switch gets overloaded, it also practices admission control by not giving dial tones.

Congestion Control in Datagram Subnets

Let us now turn to some approaches that can be used in datagram subnets (and also in virtual-circuit subnets). Each router can easily monitor the utilization of its output lines and other resources. For example, it can associate with each line a real variable, u, whose value, between 0.0 and 1.0, reflects the recent utilization of that line. To maintain a good estimate of u, a sample of the

instantaneous line utilization, f (either 0 or 1), can be made periodically and u updated according to

$$u_{new} = au_{old} + (1 - a)f$$

Where the constant a determines how fast the router forgets recent history.

Whenever u moves above the threshold, the output line enters a "warning" state. Each newly-arriving packet is checked to see if its output line is in warning state. If it is, some action is taken.

The Warning Bit

The old DECNET architecture signaled the warning state by setting a special bit in the packet's header. So does frame relay. When the packet arrived at its destination, the transport entity copied the bit into the next acknowledgement sent back to the source. The source then cut back on traffic.

As long as the router was in the warning state, it continued to set the warning bit, which meant that the source continued to get acknowledgements with it set. The source monitored the fraction of acknowledgements with the bit set and adjusted its transmission rate accordingly. As long as the warning bits continued to flow in, the source continued to decrease its transmission rate. When they slowed to a trickle, it increased its transmission rate. Note that since every router along the path could set the warning bit, traffic increased only when no router was in trouble.

Choke Packets

It uses a roundabout means to tell the source to slow down. Why not just tell it directly? In this approach, the router sends a **choke packet** back to the source host, giving it the destination found in the packet. The original packet is tagged (a header bit is turned on) so that it will not generate any more choke packets farther along the path and is then forwarded in the usual way.

Hop-by-Hop Choke Packets

At high speeds or over long distances, sending a choke packet to the source hosts does not work well because the reaction is so slow. Consider, for example, a host in San Francisco (router A) that is sending traffic to a host in New York (router D) at 155 Mbps. If the New York host begins to run out of buffers, it will take about 30 msec for a choke packet to get back to San Francisco to tell it to

slow down. The choke packet propagation is shown as the second, third, and fourth steps .In those 30 msec, another 4.6 megabits will have been sent. Even if the host in San Francisco completely shuts down immediately, the 4.6 megabits in the pipe will continue to pour in and have to be dealt with. Only in the seventh diagram will the New York router notice a slower flow.

Load Shedding

When none of the above methods make the congestion disappear, routers can bring out the heavy artillery: load shedding. **Load shedding** is a fancy way of saying that when routers are being inundated by packets that they cannot handle, they just throw them away. The term comes from the world of electrical power generation, where it refers to the practice of utilities intentionally blacking out certain areas to save the entire grid from collapsing on hot summer days when the demand for electricity greatly exceeds the supply.

A router drowning in packets can just pick packets at random to drop, but usually it can do better than that. Which packet to discard may depend on the applications running. For file transfer, an old packet is worth more than a new one because dropping packet 6 and keeping packets 7 through 10 will cause a gap at the receiver that may force packets 6 through 10 to be retransmitted (if the receiver routinely discards out-of-order packets). In a 12-packet file, dropping 6 may require 7 through 12 to be retransmitted, whereas dropping 10 may require only 10 through 12 to be retransmitted. In contrast, for multimedia, a new packet is more important than an old one. The former policy (old is better than new) is often called **wine** and the latter (new is better than old) is often called **milk**.

Random Early Detection

It is well known that dealing with congestion after it is first detected is more effective than letting it gum up the works and then trying to deal with it. This observation leads to the idea of discarding packets before all the buffer space is really exhausted. A popular algorithm for doing this is called **RED** (**Random Early Detection**) (Floyd and Jacobson, 1993). In some transport protocols (including TCP), the response to lost packets is for the source to slow down. The reasoning behind this logic is that TCP was designed for wired networks and wired networks are very reliable, so lost packets are mostly due to buffer overruns rather than transmission errors. This fact can be exploited to help reduce congestion.

Jitter Control

For applications such as audio and video streaming, it does not matter much if the packets take 20 msec or 30 msec to be delivered, as long as the transit time is constant. The variation (i.e., standard deviation) in the packet arrival times is called **jitter**. High jitter, for example, having some packets taking 20 msec and others taking 30 msec to arrive will give an uneven quality to the sound or movie. In contrast, an agreement that 99 percent of the packets be delivered with a delay in the range of 24.5 msec to 25.5 msec might be acceptable.

The range chosen must be feasible, of course. It must take into account the speed-of-light transit time and the minimum delay through the routers and perhaps leave a little slack for some inevitable delays.

The jitter can be bounded by computing the expected transit time for each hop along the path. When a packet arrives at a router, the router checks to see how much the packet is behind or ahead of its schedule. This information is stored in the packet and updated at each hop. If the packet is ahead of schedule, it is held just long enough to get it back on schedule. If it is behind schedule, the router tries to get it out the door quickly.

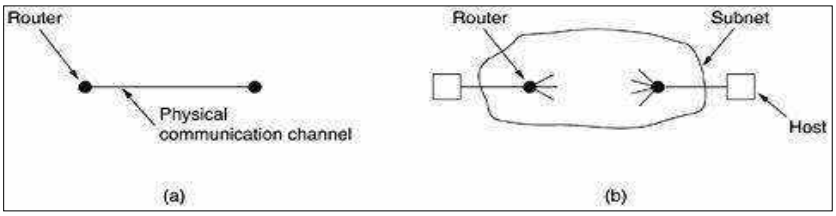
In fact, the algorithm for determining which of several packets competing for an output line should go next can always choose the packet furthest behind in its schedule. In this way,

Elements of Transport Protocols

The transport service is implemented by a **transport protocol** used between the two transport entities. In some ways, transport protocols resemble the data link protocols. Both have to deal with error control, sequencing, and flow control, among other issues.

(a) Environment of the data link layer.

(b) Environment of the transport layer



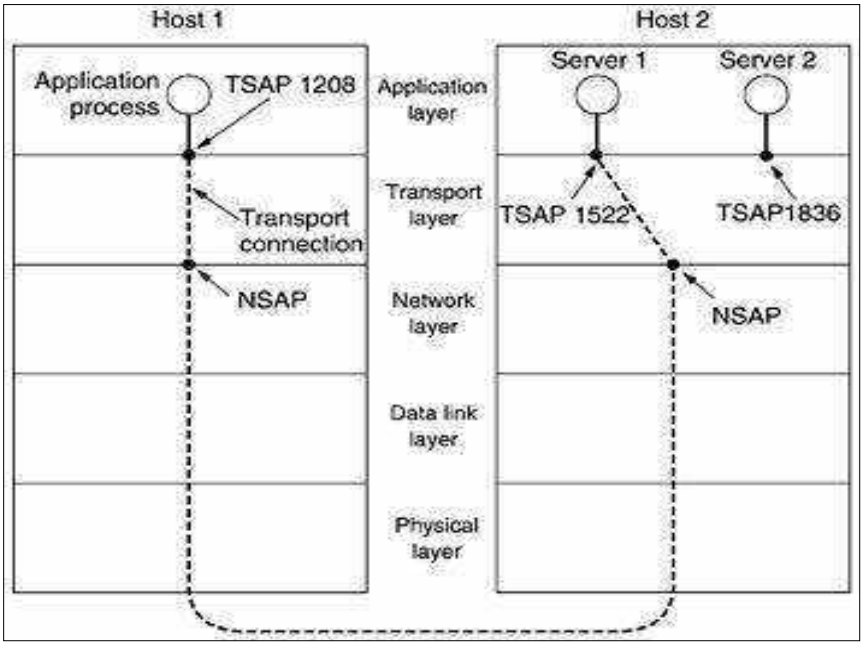
However, significant differences between the two also exist. These differences are due to major dissimilarities between the environments in which the two protocols operate, At the data link layer, two routers communicate directly via a physical channel, whereas at the transport layer, this physical channel is replaced by the entire subnet. This difference has many important implications for the protocols, as we shall see in this chapter.

TSAPs, NSAPs, and transport connections.

Addressing

When an application (e.g., a user) process wishes to set up a connection to a remote application process, it must specify which one to connect to. (Connectionless transport has the same problem: To whom should each message be sent?) The method normally used is to define transport addresses to which processes can listen for connection requests. In the Internet, these end points are called **ports**. In ATM networks, they are called **AAL-SAPs**. We will use the generic term **TSAP**, (**Transport Service Access Point**). The analogous end points in the network layer (i.e., network layer addresses) are then called **NSAPs**. IP addresses are examples of NSAPs.

TSAPs, NSAPs, and transport connections



Figure, the relationship between the NSAP, TSAP and transport connection. Application processes, both clients and servers, can attach themselves to a TSAP to establish a connection to a remote TSAP. These connections run through NSAPs on each host, as shown. The purpose of having TSAPs is that in some networks, each computer has a single NSAP, so some way is needed to distinguish multiple transport end points that share that NSAP.

A possible scenario for a transport connection is as follows:

A time of day server process on host 2 attaches itself to TSAP 1522 to wait for an incoming call. How a process attaches itself to a TSAP is outside the networking model and depends entirely on the local operating system. A call such as our LISTEN might be used, for example.

An application process on host 1 wants to find out the time-of-day, so it issues a CONNECT request specifying TSAP 1208 as the source and TSAP 1522 as the destination. This action ultimately results in a transport connection being established between the application process on host 1 and server 1 on host 2.

- The application process then sends over a request for the time.
- The time server process responds with the current time.

The transport connection is then released. Note that there may well be other servers on host 2 that are attached to other TSAPs and waiting for incoming connections that arrive over the same NSAP.

The picture painted above is fine, except we have swept one little problem under the rug: How does the user process on host 1 know that the time-of-day server is attached to TSAP 1522? One possibility is that the time-of-day server has been attaching itself to TSAP 1522 for years and gradually all the network users have learned this. In this model, services have stable TSAP addresses that are listed in files in well-known places, such as the /etc/services file on UNIX systems, which lists which servers are permanently attached to which ports.

Connection Establishment

Establishing a connection sounds easy, but it is actually surprisingly tricky. At first glance, it would seem sufficient for one transport entity to just send a CONNECTION REQUEST TPDU to the destination and wait for a

CONNECTION ACCEPTED reply. The problem occurs when the network can lose, store, and duplicate packets. This behavior causes serious complications.

Imagine a subnet that is so congested that acknowledgements hardly ever get back in time and each packet times out and is retransmitted two or three times. Suppose that the subnet uses datagrams inside and that every packet follows a different route. Some of the packets might get stuck in a traffic jam inside the subnet and take a long time to arrive, that is, they are stored in the subnet and pop out much later.

Packet lifetime can be restricted to a known maximum using one (or more) of the following techniques:

- Restricted subnet design.
- Putting a hop counter in each packet.
- Time stamping each packet.

The first method includes any method that prevents packets from looping, combined with some way of bounding congestion delay over the (now known) longest possible path. The second method consists of having the hop count initialized to some appropriate value and decremented each time the packet is forwarded. The network protocol simply discards any packet whose hop counter becomes zero. The third method requires each packet to bear the time it was created, with the routers agreeing to discard any packet older than some agreed-upon time. This latter method requires the router clocks to be synchronized, which itself is a nontrivial task unless synchronization is achieved external to the network, for example by using GPS or some radio station that broadcasts the precise time periodically.

Connection Release

Releasing a connection is easier than establishing one. Nevertheless, there are more pitfalls than one might expect. As we mentioned earlier, there are two styles of terminating a connection: asymmetric release and symmetric release. Asymmetric release is the way the telephone system works: when one party hangs up, the connection is broken. Symmetric release treats the connection as two separate unidirectional connections and requires each one to be released separately.

Asymmetric release is abrupt and may result in data loss. After the connection is established, host 1 sends a TPDU that arrives properly at host 2.

Then host 1 sends another TPDU. Unfortunately, host 2 issues a DISCONNECT before the second TPDU arrives. The result is that the connection is released and data are lost.

Flow Control and Buffering

Having examined connection establishment and release in some detail, let us now look at how connections are managed while they are in use. One of the key issues has come up before: flow control. In some ways the flow control problem in the transport layer is the same as in the data link layer, but in other ways it is different. The basic similarity is that in both layers a sliding window or other scheme is needed on each connection to keep a fast transmitter from overrunning a slow receiver. The main difference is that a router usually has relatively few lines, whereas a host may have numerous connections. This difference makes it impractical to implement the data link buffering strategy in the transport layer.

In protocol 6, for example, both sender and receiver are required to dedicate MAX_SEQ + 1 buffers to each line, half for input and half for output. For a host with a maximum of, say, 64 connections, and a 4-bit sequence number, this protocol would require 1024 buffers.

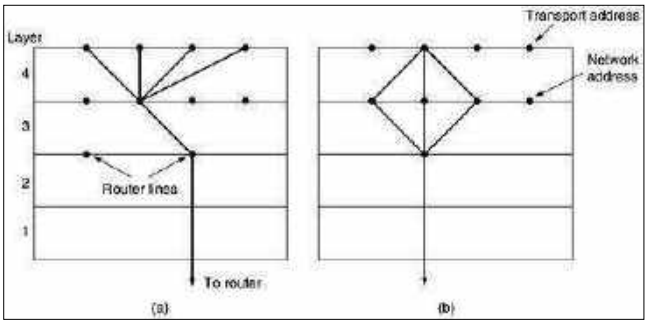
In the data link layer, the sending side must buffer outgoing frames because they might have to be retransmitted. If the subnet provides datagram service, the sending transport entity must also buffer, and for the same reason. If the receiver knows that the sender buffers all TPDUs until they are acknowledged, the receiver may or may not dedicate specific buffers to specific connections, as it sees fit. The receiver may, for example, maintain a single buffer pool shared by all connections. When a TPDU comes in, an attempt is made to dynamically acquire a new buffer. If one is available, the TPDU is accepted; otherwise, it is discarded. Since the sender is prepared to retransmit TPDUs lost by the subnet, no harm is done by having the receiver drop TPDUs, although some resources are wasted. The sender just keeps trying until it gets an acknowledgement.

Multiplexing

Multiplexing several conversations onto connections, virtual circuits, and physical links plays a role in several layers of the network architecture. In the transport layer the need for multiplexing can arise in a number of ways. For example, if only one network address is available on a host, all transport

connections on that machine have to use it. When a TPDU comes in, some way is needed to tell which process to give it to. This situation, called **upward multiplexing**. In this figure, four distinct transport connections alluse the same network connection (e.g., IP address) to the remote host.

(a) Upward multiplexing
(b) Downward multiplexing



Multiplexing can also be useful in the transport layer for another reason. Suppose, for example, that a subnet uses virtual circuits internally and imposes a maximum data rate on each one. If a user needs more bandwidth than one virtual circuit can provide, a way out is to open multiple network connections and distribute the traffic among them on a round-robin basis. This modus operandi is called **downward multiplexing**. With k network connections open, the effective bandwidth is increased by a factor of k. A common example of downward multiplexing occurs with home users who have an ISDN line. This line provides for two separate connections of 64 kbps each. Using both of them to call an Internet provider and dividing the traffic over both lines makes it possible to achieve an effective bandwidth of 128 kbps.

Crash Recovery

If hosts and routers are subject to crashes, recovery from these crashes becomes an issue. If the transport entity is entirely within the hosts, recovery from network and router crashes is straightforward. If the network layer provides datagram service, the transport entities expect lost TPDUs all the time and know how to cope with them. If the network layer provides connection-oriented service, then loss of a virtual circuit is handled by establishing a new one and then probing the remote transport entity to ask it which TPDUs it has received and which ones it has not received. The latter ones can be retransmitted.

A more troublesome problem is how to recover from host crashes. In particular, it may be desirable for clients to be able to continue working when servers crash and then quickly reboot. To illustrate the difficulty, let us assume that one host, the client, is sending a long file to another host, the file server, using a simple stop-and-wait protocol. The transport layer on the server simply passes the incoming TPDUs to the transport user, one by one. Partway through the transmission, the server crashes. When it comes back up, its tables are reinitialized, so it no longer knows precisely where it was.

In an attempt to recover its previous status, the server might send a broadcast TPDU to all other hosts, announcing that it had just crashed and requesting that its clients inform it of the status of all open connections. Each client can be in one of two states: one TPDU outstanding, S1, or no TPDUs outstanding, S0. Based on only this state information, the client must decide whether to retransmit the most recent TPDU.

The Internet Transport Protocols: TCP

Introduction to TCP

TCP (**Transmission Control Protocol**) was specifically designed to provide a reliable end-to-end byte stream over an unreliable internetwork. An internetwork differs from a single network because different parts may have wildly different topologies, bandwidths, delays, packet sizes, and other parameters. TCP was designed to dynamically adapt to properties of the internetwork and to be robust in the face of many kinds of failures.

TCP was formally defined in RFC 793. As time went on, various errors and inconsistencies were detected, and the requirements were changed in some areas. These clarifications and some bug fixes are detailed in RFC 1122. Extensions are given in RFC 1323.

Each machine supporting TCP has a TCP transport entity, either a library procedure, a user process, or part of the kernel. In all cases, it manages TCP streams and interfaces to the IP layer. A TCP entity accepts user data streams from local processes, breaks them up into pieces not exceeding 64 KB (in practice, often 1460 data bytes in order to fit in a single Ethernet frame with the IP and TCP headers), and sends each piece as a separate IP datagram. When datagrams containing TCP data arrive at a machine, they are given to the TCP entity, which reconstructs the original byte streams. For simplicity, we will

sometimes use just "TCP" to mean the TCP transport entity (a piece of software) or the TCP protocol (a set of rules). From the context it will be clear which is meant. For example, in "The user gives TCP the data," the TCP transport entity is clearly intended.

The IP layer gives no guarantee that datagrams will be delivered properly, so it is up to TCP to time out and retransmit them as need be. Datagrams that do arrive may well do so in the wrong order; it is also up to TCP to reassemble them into messages in the proper sequence. In short, TCP must furnish the reliability that most users want and that IP does not provide.

The TCP Service Model

TCP service is obtained by both the sender and receiver creating end points, called sockets, Each socket has a socket number (address) consisting of the IP address of the host and a 16-bit number local to that host, called a **port**. A port is the TCP name for a TSAP. For TCP service to be obtained, a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine. A socket may be used for multiple connections at the same time. In other words, two or more connections may terminate at the same socket. Connections are identified by the socket identifiers at both ends that is, (socket1, socket2). No virtual circuit numbers or other identifiers are used.

Port numbers below 1024 are called **well-known ports** and are reserved for standard services. For example, any process wishing to establish a connection to a host to transfer a file using FTP can connect to the destination host's port 21 to contact its FTP daemon.

It would certainly be possible to have the FTP daemon attach itself to port 21 at boot time, the telnet daemon to attach itself to port 23 at boot time, and so on. However, doing so would clutter up memory with daemons that were idle most of the time. Instead, what is generally done is to have a single daemon, called **inetd** (**Internet daemon**) in UNIX, attach itself to multiple ports and wait for the first incoming connection. When that occurs, inetd forks off a new process and executes the appropriate daemon in it, letting that daemon handle the request. In this way, the daemons other than inetd are only active when there is work for them to do. Inetd learns which ports it is to use from a configuration file. Consequently, the system administrator can set up the system to have permanent daemons on the busiest ports (e.g., port 80) and inetd on the rest.

All TCP connections are full duplex and point-to-point. Full duplex means that traffic can go in both directions at the same time. Point-to-point means that each connection has exactly two end points. TCP does not support multicasting or broadcasting.

A TCP connection is a byte stream, not a message stream. Message boundaries are not preserved end to end. For example, if the sending process does four 512-byte writes to a TCP stream, these data may be delivered to the receiving process as four 512-byte chunks, two 1024-byte chunks, one 2048-byte chunk.

The TCP Protocol

A key feature of TCP, and one which dominates the protocol design, is that every byte on a TCP connection has its own 32-bit sequence number. When the Internet began, the lines between routers were mostly 56-kbps leased lines, so a host blasting away at full speed took over 1 week to cycle through the sequence numbers. At modern network speeds, the sequence numbers can be consumed at an alarming rate, as we will see later. Separate 32-bit sequence numbers are used for acknowledgements and for the window mechanism, as discussed below.

The sending and receiving TCP entities exchange data in the form of segments. A **TCP segment** consists of a fixed 20-byte header (plus an optional part) followed by zero or more data bytes. The TCP software decides how big segments should be. It can accumulate data from several writes into one segment or can split data from one write over multiple segments. Two limits restrict the segment size. First, each segment, including the TCP header, must fit in the 65,515-byte IP payload. Second, each network has a **maximum transfer unit**, or **MTU**, and each segment must fit in the MTU. In practice, the MTU is generally 1500 bytes (the Ethernet payload size) and thus defines the upper bound on segment size.

The basic protocol used by TCP entities is the sliding window protocol. When a sender transmits a segment, it also starts a timer. When the segment arrives at the destination, the receiving TCP entity sends back a segment (with data if any exist, otherwise without data) bearing an acknowledgement number equal to the next sequence number it expects to receive. If the sender's timer goes off before the acknowledgement is received, the sender transmits the segment again.

The TCP Segment Header

Every segment begins with a fixed-format,20-byte header. The fixed header may be followed by header options. After the options, if any, up to 65,535 - 20 - 20 = 65,495 data bytes may follow, where the first 20 refer to the IP header and the second to the TCP header. Segments without any data are legal and are commonly used for acknowledgements and control messages.

Let us dissect the TCP header field by field. The Source port and Destination port fields identify the local end points of the connection. A port plus its host's IP address forms a 48-bit unique end point. The source and destination end points together identify the connection.

The Sequence number and Acknowledgement number fields perform their usual functions. Note that the latter specifies the next byte expected, not the last byte correctly received. Both are 32 bits long because every byte of data is numbered in a TCP stream.

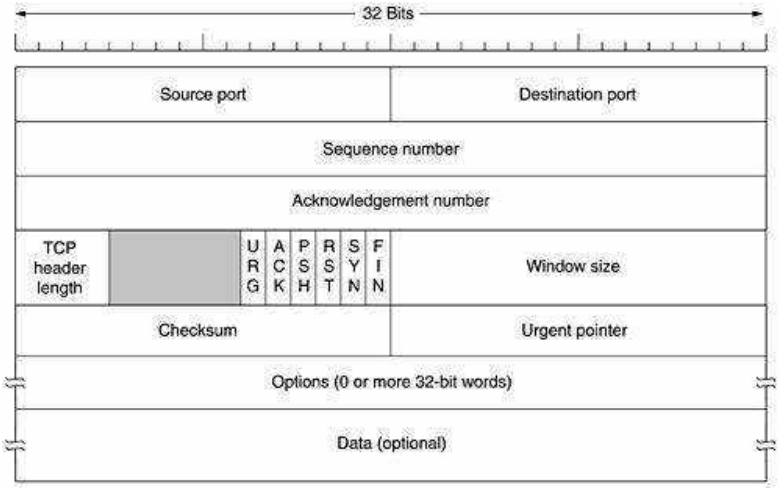
The TCP header length tells how many 32-bit words are contained in the TCP header. This information is needed because the Options field is of variable length, so the header is, too. Technically, this field really indicates the start of the data within the segment, measured in 32-bit words, but that number is just the header length in words, so the effect is the same.

Next comes a 6-bit field that is not used. The fact that this field has survived intact for over a quarter of a century is testimony to how well thought out TCP is. Lesser protocols would have needed it to fix bugs in the original design. Now come six 1-bit flags. URG is set to 1 if the Urgent pointer is in use. The Urgent pointer is used to indicate a byte offset from the current sequence number at which urgent data are to be found. This facility is in lieu of interrupt messages. As we mentioned above, this facility is a bare-bones way of allowing the sender to signal the receiver without getting TCP itself involved in the reason for the interrupt.

The ACK bit is set to 1 to indicate that the Acknowledgement number is valid. If ACK is 0, the segment does not contain an acknowledgement so the Acknowledgement number field is ignored.

The PSH bit indicates PUSHED data. The receiver is hereby kindly requested to deliver the data to the application upon arrival and not buffer it until a full buffer has been received (which it might otherwise do for efficiency).

The TCP header



The RST bit is used to reset a connection that has become confused due to a host crash or some other reason. It is also used to reject an invalid segment or refuse an attempt to open a connection. In general, if you get a segment with the RST bit on, you have a problem on your hands.

The SYN bit is used to establish connections. The connection request has SYN = 1 and ACK = 0 to indicate that the piggyback acknowledgement field is not in use. The connection reply does bear an acknowledgement, so it has SYN = 1 and ACK = 1. In essence the SYN bit is used to denote CONNECTION REQUEST and CONNECTION ACCEPTED, with the ACK bit used to distinguish between those two possibilities.

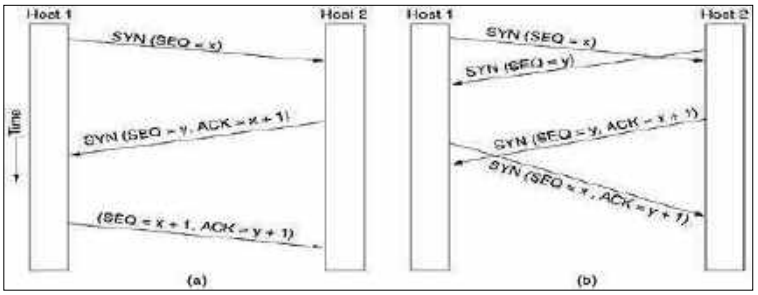
The FIN bit is used to release a connection. It specifies that the sender has no more data to transmit. However, after closing a connection, the closing process may continue to receive data indefinitely. Both SYN and FIN segments have sequence numbers and are thus guaranteed to be processed in the correct order.

TCP Connection Establishment

Connections are established in TCP by means of the three-way handshake discussed in Sec.6.2.2. To establish a connection, one side, say, the server,

passively waits for an incoming connection by executing the LISTEN and ACCEPT primitives, either specifying a specific source or nobody in particular.

(a) TCP connection establishment in the normal case. (b) Call collision



The other side, say, the client, executes a CONNECT primitive, specifying the IP address and port to which it wants to connect, the maximum TCP segment size it is willing to accept, and optionally some user data (e.g., a password). The CONNECT primitive sends a TCP segment with the SYN bit on and ACK bit off and waits for a response.

When this segment arrives at the destination, the TCP entity there checks to see if there is a process that has done a LISTEN on the port given in the Destination port field. If not, it sends a reply with the RST bit on to reject the connection.

If some process is listening to the port, that process is given the incoming TCP segment. It can then either accept or reject the connection. If it accepts, an acknowledgement segment is sent back. The Note that a SYN segment consumes 1 byte of sequence space so that it can be acknowledged unambiguously.

In the event that two hosts simultaneously attempt to establish a connection between the same two sockets. The result of these events is that just one connection is established, not two because connections are identified by their end points. If the first setup results in a connection identified by (x, y) and the second one does too, only one table entry is made, namely, for (x, y).

The initial sequence number on a connection is not 0 for the reasons we discussed earlier. A clock-based scheme is used, with a clock tick every 4 µsec. For additional safety, when a host crashes, it may not reboot for the maximum packet lifetime to make sure that no packets from previous connections are still roaming around the Internet somewhere.

UNIT - V
APPLICATION LAYER

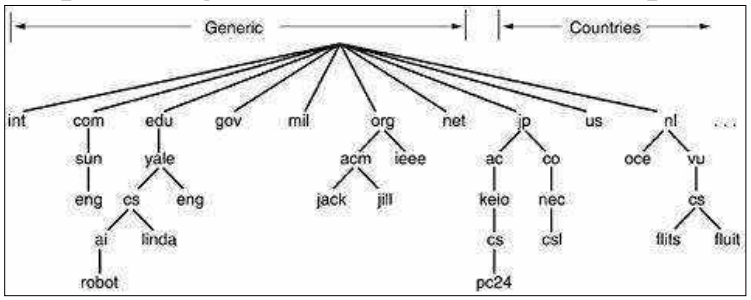
DNS— The Domain Name System

However, when thousands of minicomputers and PCs were connected to the net, everyone realized that this approach could not continue to work forever. For one thing, the size of the file would become too large. However, even more important, host name conflicts would occur constantly unless names were centrally managed, something unthinkable in a huge international network due to the load and latency. To solve these problems, **DNS** (the **Domain Name System**) was invented.

The essence of DNS is the invention of a hierarchical, domain-based naming scheme and a distributed database system for implementing this naming scheme. It is primarily used for mapping host names and e-mail destinations to IP addresses but can also be used for other purposes. DNS is defined in RFCs 1034 and 1035.

Very briefly, the way DNS is used is as follows. To map a name onto an IP address, an application program calls a library procedure called the **resolver**, passing it the name as a parameter. The resolver sends a UDP packet to a local DNS server, which then looks up the name and returns the IP address to the resolver, which then returns it to the caller. Armed with the IP address, the program can then establish a TCP connection with the destination or send it UDP packets.

A portion of the Internet domain name space



The DNS Name Space

Managing a large and constantly changing set of names is a nontrivial problem. In the postal system, name management is done by requiring letters to specify (implicitly or explicitly) the country, state or province, city, and street address of the addressee. By using this kind of hierarchical addressing, there is no confusion between the Marvin Anderson on Main St. in White Plains, N.Y. and the Marvin Anderson on Main St. in Austin, Texas. DNS works the same way.

Conceptually, the Internet is divided into over 200 top-level **domains**, where each domain covers many hosts. Each domain is partitioned into subdomains, and these are further partitioned, and so on. The leaves of the tree represent domains that have no subdomains (but do contain machines, of course). A leaf domain may contain a single host, or it may represent a company and contain thousands of hosts.

The top-level domains come in two flavors: generic and countries. The original generic domains were com (commercial), edu (educational institutions), gov (the U.S. Federal Government), int (certain international organizations), mil (the U.S. armed forces), net (network providers), and org (nonprofit organizations). The country domains include one entry for every country, as defined in ISO 3166.

Each domain is named by the path upward from it to the (unnamed) root. The components are separated by periods (pronounced "dot"). Thus, the engineering department at Sun Microsystems might be example: sun.com., rather than a UNIX-style name such as /com/sun/eng. Notice that this hierarchical naming means that example: sun.com.does not conflict with a potential use of eng in eng.yale.edu., which might be used by the Yale English department.

Domain names can be either absolute or relative. An absolute domain name always ends with a period (e.g., eng.sun.com.), whereas a relative one does not. Relative names have to be interpreted in some context to uniquely determine their true meaning. In both cases, a named domain refers to a specific node in the tree and all the nodes under it. Domain names are case insensitive, so edu, Edu, and EDU mean the same thing. Component names can be up to 63 characters long, and full path names must not exceed 255 characters.

Each domain controls how it allocates the domains under it. For example, Japan has domains ac.jp and co.jp that mirroreduand com. The Netherlands does not make this distinction and puts all organizations directly under nl.

Thus, all three of the following are university computer science departments:

- cs.yale.edu (Yale University, in the United States)
- cs.vu.nl (VrijeUniversiteit, in The Netherlands)
- cs.keio.ac.jp (Keio University, in Japan)

To create a new domain, permission is required of the domain in which it will be included. For example, if a VLSI group is started at Yale and wants to be known as vlsi.cs.yale.edu, it has to get permission from whoever manages cs.yale.edu. Similarly, if a new university is chartered, say, the University of Northern South Dakota, it must ask the manager of the edu domain to assign it unsd.edu. In this way, name conflicts are avoided and each domain can keep track of all its subdomains. Once a new domain has been created and registered, it can create subdomains, such as cs.unsd.edu, without getting permission from anybody higher up the tree.

Resource Records

Every domain, whether it is a single host or a top-level domain, can have a set of **resource records** associated with it. For a single host, the most common resource record is just its IP address, but many other kinds of resource records also exist. When a resolver gives a domain name to DNS, what it gets back are the resource records associated with that name. Thus, the primary function of DNS is to map domain names onto resource records.

The principal DNS resource record types for IPv4

Type	Meaning	Value
SOA	Start of Authority	Parameters for this zone
A	IP address of a host	32-Bit integer
MX	Mail exchange	Priority, domain willing to accept e-mail
NS	Name Server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
HINFO	Host description	CPU and OS in ASCII
TXT	Text	Uninterpreted ASCII text

A resource record is a five-tuple. Although they are encoded in binary for efficiency, in most expositions, resource records are presented as ASCII text, one line per resource record. The format we will use is as follows:

Domain _name Time _ to _live Class Type Value

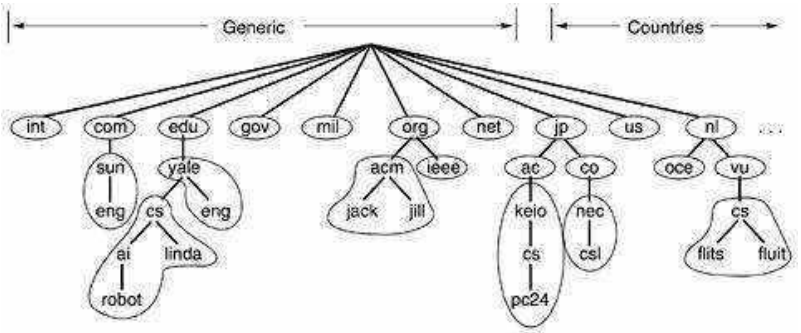
The Domain _name tells the domain to which this record applies. Normally, many records exist for each domain and each copy of the database holds information about multiple domains. This field is thus the primary search key used to satisfy queries. The order of the records in the database is not significant.

The Time _ to _live field gives an indication of how stable the record is. Information that is highly stable is assigned a large value, such as 86400 (the number of seconds in 1 day). Information that is highly volatile is assigned a small value, such as 60 (1 minute). We will come back to this point later when we have discussed caching.

The third field of every resource record is the Class. For Internet information, it is always IN. For non-Internet information, other codes can be used, but in practice, these are rarely seen.

An SOA record provides the name of the primary source of information about the name server's zone (described below), the e-mail address of its administrator, a unique serial number, and various flags and timeouts.

The most important record type is the A (Address) record. It holds a 32-bit IP address for some host. Every Internet host must have at least one IP address so that other machines can communicate with it. Some hosts have two or more network connections, in which case they will have one type A resource record per network connection (and thus per IP address). DNS can be configured to cycle through these, returning the first record on the first request, the second record on the second request, and so on.



Part of the DNS name space showing the division into zones

Name Servers

In theory at least, a single name server could contain the entire DNS database and respond to all queries about it. In practice, this server would be so overloaded as to be useless. Furthermore, if it ever went down, the entire Internet would be crippled.

To avoid the problems associated with having only a single source of information, the DNS name space is divided into non-overlapping **zones**. Each zone contains some part of the tree and also contains name servers holding the information about that zone. Normally, a zone will have one primary name server, which gets its information from a file on its disk, and one or more secondary name servers, which get their information from the primary name server. To improve reliability, some servers for a zone can be located outside the zone.

Where the zone boundaries are placed within a zone is up to that zone's administrator. This decision is made in large part based on how many name servers are desired, and where. Yale has a server for yale.edu that handles eng.yale.edu but not cs.yale.edu, which is a separate zone with its own name servers. Such a decision might bemade when a department such as English does not wish to run its own name server, but a department such as computer science does. Consequently, cs.yale.edu is a separate zone but eng.yale.edu is not.

When a resolver has a query about a domain name, it passes the query to one of the local name servers. If the domain being sought falls under the jurisdiction of the name server, such as ai.cs.yale.edu falling under cs.yale.edu, it returns the authoritative resource records. An **authoritative record** is one that

comes from the authority that manages the record and isthus always correct. Authoritative records are in contrast to cached records, which may be out of date.

While DNS is extremely important to the correct functioning of the Internet, all it really does is map symbolic names for machines onto their IP addresses. It does not help locate people, resources, services, or objects in general. For locating these things, another directory service has been defined, called **LDAP** (**Lightweight Directory Access Protocol**). It is a simplified version of the OSI X.500 directory service and is described in RFC 2251. It organizes information as a tree and allows searches on different components. It can be regarded as a "white pages" telephone book. We will not discuss it further in this book, but for more information see (Weltman and Dahbura, 2000).

Electronic Mail

Electronic mail, or **e-mail**, as it is known to its many fans, has been around for over two decades. Before 1990, it was mostly used in academia. During the 1990s, it became known to the public at large and grew exponentially to the point where the number of e-mails sent per day now is vastly more than the number of **snail mail** (i.e., paper) letters.

Some smileys. They will not be on the final exam

Smiley	Meaning	Smiley	Meaning	Smiley	Meaning
:-)	I'm happy	=I:-)	Abe Lincoln	:+)	Big nose
:-(	I'm sad/angry	=):-)	Uncle Sam	:-))	Double chin
:-I	I'm apathetic	*<:-)	Santa Claus	:-{)	Mustache
;-)	I'm winking	<:-(	Dunce	#:-)	Matted hair
:-(O)	I'm yelling	(-:	Australian	8-)	Wears glasses
:-(*)	I'm vomiting	:-)X	Man with bowtie	C:-)	Large brain

E-mail, like most other forms of communication, has its own conventions and styles. In particular, it is very informal and has a low threshold of use. People who would never dream of calling up or even writing a letter to a Very Important Person do not hesitate for a second to send a sloppily-written e-mail.

E-mail is full of jargon such as BTW (By The Way), ROTFL (Rolling On The Floor Laughing), and IMHO (In My Humble Opinion). Many people also use little ASCII symbols called **smileys** or **emoticons** in their e-mail. A few of the more interesting ones are reproduced inFig. 7-6. Foremost, rotating the book

90 degrees clockwise will make them clearer. For a mini book giving over 650 smileys, see (Sanderson and Dougherty, 1993).

The first e-mail systems simply consisted of file transfer protocols, with the convention that the first line of each message (i.e., file) contained the recipient's address. As time went on, the limitations of this approach became more obvious.

Some of the complaints were as follows:

- Sending a message to a group of people was inconvenient. Managers often need this facility to send memos to all their subordinates.
- Messages had no internal structure, making computer processing difficult. For example, if a forwarded message was included in the body of another message, extracting the forwarded part from the received message was difficult.
- The originator (sender) never knew if a message arrived or not.
- If someone was planning to be away on business for several weeks and wanted all incoming e-mail to be handled by his secretary, this was not easy to arrange.
- The user interface was poorly integrated with the transmission system requiring users first to edit a file, then leave the editor and invoke the file transfer program.
- It was not possible to create and send messages containing a mixture of text, drawings, facsimile, and voice.

Architecture and Services

They normally consist of two subsystems: the **user agents**, which allow people to read and send e-mail, and the **message transfer agents**, which move the messages from the source to the destination. The user agents are local programs that provide a command-based, menu-based, or graphical method for interacting with the e-mail system. The message transfer agents are typically system **daemons**, that is, processes that run in the background. Their job is to move e-mail through the system.

Typically, e-mail systems support five basic functions. Let us take a look at them.

➢ **Composition** refers to the process of creating messages and answers. Although any text editor can be used for the body of the message, the system itself can provide assistance with addressing and the numerous header fields

attached to each message. For example, when answering a message, the e-mail system can extract the originator's address from the incoming e-mail and automatically insert it into the proper place in the reply.

- **Transfer** refers to moving messages from the originator to the recipient. In large part, this requires establishing a connection to the destination or some intermediate machine, outputting the message, and releasing the connection. The e-mail system should do this automatically, without bothering the user.
- **Reporting** has to do with telling the originator what happened to the message. Was it delivered? Was it rejected? Was it lost? Numerous applications exist in which confirmation of delivery is important and may even have legal significance ("Well, Your Honor, my e-mail system is not very reliable, so I guess the electronic subpoena just got lost somewhere").
- **Displaying** incoming messages is needed so people can read their e-mail. Sometimes conversion is required or a special viewer must be invoked, for example, if the message is a PostScript file or digitized voice. Simple conversions and formatting are sometimes attempted as well.
- **Disposition** is the final step and concerns what the recipient does with the message after receiving it. Possibilities include throwing it away before reading, throwing it away after reading, saving it, and so on. It should also be possible to retrieve and reread saved messages, forward them, or process them in other ways.

Envelopes and messages. (a) Paper mail. (b) Electronic mail

In addition to these basic services, some e-mail systems, especially internal corporate ones, provide a variety of advanced features. Let us just briefly mention a few of these. When people move or when they are away for some period of time, they may want their e-mail forwarded, so the system should be able to do this automatically.

Most systems allow users to create **mailboxes** to store incoming e-mail. Commands are needed to create and destroy mailboxes, inspect the contents of mailboxes, insert and delete messages from mailboxes, and so on.

Corporate managers often need to send a message to each of their subordinates, customers, or suppliers. This gives rise to the idea of a **mailing list**, which is a list of e-mail addresses. When a message is sent to the mailing list, identical copies are delivered to everyone on the list.

Other advanced features are carbon copies, blind carbon copies, high-priority e-mail, secret (i.e., encrypted) e-mail, alternative recipients if the primary one is not currently available, and the ability for secretaries to read and answer their bosses' e-mail.

E-mail is now widely used within industry for intracompany communication. It allows far-flung employees to cooperate on complex projects, even over many time zones. By eliminating most cues associated with rank, age, and gender, e-mail debates tend to focus on ideas, not on corporate status. With e-mail, a brilliant idea from a summer student can have more impact than a dumb one from an executive vice president.

A key idea in e-mail systems is the distinction between the **envelope** and its contents. The envelope encapsulates the message. It contains all the information needed for transporting the message, such as the destination address, priority, and security level, all of which are distinct from the message itself. The message transport agents use the envelope for routing, just as the post office does.

The message inside the envelope consists of two parts: the **header** and the **body**. The header contains control information for the user agents. The body is entirely for the human recipient.

The User Agent

E-mail systems have two basic parts, as we have seen: the user agents and the message transfer agents. In this section we will look at the user agents. A user agent is normally a program (sometimes called a mail reader) that accepts a variety of commands for composing, receiving, and replying to messages, as well as for manipulating mailboxes. Some user agents have a fancy menu- or icon-driven interface that requires a mouse, whereas others expect 1-character commands from the keyboard. Functionally, these are the same. Some systems are menu- or icon-driven but also have keyboard shortcuts.

Sending E-mail

To send an e-mail message, a user must provide the message, the destination address, and possibly some other parameters. The message can be produced with a free-standing text editor, a word processing program, or possibly with a specialized text editor built into the user agent. The destination address must be in a format that the user agent can deal with. Many user agents expect

addresses of the form user@dns-address. Since we have studied DNS earlier in this chapter, we will not repeat that material here.

However, it is worth noting that other forms of addressing exist. In particular, X.400 addresses look radically different from DNS addresses. They are composed of attribute = value pairs separated by slashes, for example,

/C=US/ST=MASSACHUSETTS/L=CAMBRIDGE/PA=360 MEMORIAL DR./CN=KEN SMITH/

This address specifies a country, state, locality, personal address and a common name (Ken Smith). Many other attributes are possible, so you can send e-mail to someone whose exact e-mail address you do not know, provided you know enough other attributes (e.g., company and job title). Although X.400 names are considerably less convenient than DNS names, most e-mail systems have aliases (sometimes called nicknames) that allow users to enter or select a person's name and get the correct e-mail address. Consequently, even with X.400 addresses, it is usually not necessary to actually type in these strange strings.

Most e-mail systems support mailing lists, so that a user can send the same message to a list of people with a single command. If the mailing list is maintained locally, the user agent can just send a separate message to each intended recipient. However, if the list is maintained remotely, then messages will be expanded there. For example, if a group of bird watchers has a mailing list called birders installed on meadowlark.arizona.edu, then any message sent to birders@meadowlark.arizona.edu will be routed to the University of Arizona and expandedthere into individual messages to all the mailing list members, wherever in the world they may be. Users of this mailing list cannot tell that it is a mailing list. It could just as well be the personal mailbox of Prof. Gabriel O. Birders.

Reading E-mail

Typically, when a user agent is started up, it looks at the user's mailbox for incoming e-mail before displaying anything on the screen. Then it may announce the number of messages in the mailbox or display a one-line summary of each one and wait for a command.

An example display of the contents of a mailbox

#	Flags	Bytes	Sender	Subject
1	K	1030	asw	Changes to MINIX
2	KA	6348	trudy	Not all Trudys are nasty
3	K F	4519	Amy N. Wong	Request for information
4		1236	bal	Bioinformatics
5		104110	kaashoek	Material on peer-to-peer
6		1223	Frank	Re: Will you review a grant proposal
7		3110	guido	Our paper has been accepted
8		1204	dmr	Re: My student's visit

As an example of how a user agent works, let us take a look at a typical mail scenario. After starting up the user agent, the user asks for a summary of his e-mail. Each line refers to one message. In this example, the mailbox contains eight messages.

Each line of the display contains several fields extracted from the envelope or header of the corresponding message. In a simple e-mail system, the choice of fields displayed is built into the program. In a more sophisticated system, the user can specify which fields are to be displayed by providing a **user profile**, a file describing the display format. In this basic example, the first field is the message number. The second field, Flags, can contain a K, meaning that the message is not new but was read previously and kept in the mailbox; an A, meaning that the message has already been answered; and/or an F, meaning that the message has been forwarded to someone else. Other flags are also possible.

The third field tells how long the message is, and the fourth one tells who sent the message. Since this field is simply extracted from the message, this field may contain first names, full names, initials, login names, or whatever else the sender chooses to put there. Finally, the Subject field gives a brief summary of what the message is about. People who fail to include a Subject field often discover that responses to their e-mail tend not to get the highest priority.

After the headers have been displayed, the user can perform any of several actions, such as displaying a message, deleting a message, and so on. The older systems were text based and typically used one-character commands for performing these tasks, such as T (type message), A (answer message), D (delete message), and F (forward message). An argument specified the message in question. More recent systems use graphical interfaces. Usually, the user selects

a message with the mouse and then clicks on an icon to type, answer, delete, or forward it.

E-mail has come a long way from the days when it was just file transfer. Sophisticated user agents make managing a large volume of e-mail possible. For people who receive and send thousands of messages a year, such tools are invaluable.

Message Formats

Let us now turn from the user interface to the format of the e-mail messages themselves. First we will look at basic ASCII e-mail using RFC 822. After that, we will look at multimedia extensions to RFC 822.

RFC 822 header fields related to message transport.

Header	Meaning
To:	E-mail address(es) of primary recipient(s)
Cc:	E-mail address(es) of secondary recipient(s)
Bcc:	E-mail address(es) for blind carbon copies
From:	Person or people who created the message
Sender:	E-mail address of the actual sender
Received:	Line added by each transfer agent along the route
Return-Path:	Can be used to identify a path back to the sender

RFC 822

Messages consist of a primitive envelope (described in RFC 821), some number of header fields, a blank line, and then the message body. Each header field (logically) consists of a single line of ASCII text containing the field name, a colon, and, for most fields, a value. RFC 822 was designed decades ago and does not clearly distinguish the envelope fields from the header fields. Although it was revised in RFC 2822, completely redoing it was not possible due to its widespread usage. In normal usage, the user agent builds a message and passes it to the message transfer agent, which then uses some of the header fields to construct the actual envelope, a somewhat old-fashioned mixing of message and envelope.

The principal header fields related to message transport are listed. The field gives the DNS address of the primary recipient. Having multiple recipients is also

allowed. The Cc: field gives the addresses of any secondary recipients. In terms of delivery, there is no distinction between the primary and secondary recipients. It is entirely a psychological difference that may be important to the people involved but is not important to the mail system. The term Cc: (Carbon copy) is a bit dated, since computers do not use carbon paper, but it is well established. The Bcc: (Blind carbon copy) field is like the Cc: field, except that this line is deleted from all the copies sent to the primary and secondary recipients. This feature allows people to send copies to third parties without the primary and secondary recipients knowing this.

Some fields used in the RFC 822 message header.

Header	Meaning
Date:	The date and time the message was sent
Reply-To:	E-mail address to which replies should be sent
Message-Id:	Unique number for referencing this message later
In-Reply-To:	Message-Id of the message to which this is a reply
References:	Other relevant Message-Ids
Keywords:	User-chosen keywords
Subject:	Short summary of the message for the one-line display

The next two fields, from: and Sender: tell who wrote and sent the message, respectively. These need not be the same. For example, a business executive may write a message, but her secretary may be the one who actually transmits it. In this case, the executive would be listed in the from: field and the secretary in the Sender: field. The From: field is required, but the Sender: field may be omitted if it is the same as the From: field. These fields are needed in case the message is undeliverable and must be returned to the sender.

A line containing Received: is added by each message transfer agent along the way. The line contains the agent's identity, the date and time the message was received, and other information that can be used for finding bugs in the routing system.

The Return-Path: field is added by the final message transfer agent and was intended to tell how to get back to the sender. In theory, this information can be gathered from all the Received: headers (except for the name of the sender's mailbox), but it is rarely filled in assuch and typically just contains the sender's address.

RFC 822 messages may also contain a variety of header fields used by the user agents or human recipients. The most common ones are listed in Fig.7-10. Most of these are self-explanatory, so we will not go into all of them in detail.

The Reply-To: field is sometimes used when neither the person composing the message nor the person sending the message wants to see the reply. For example, a marketing manager writes an e-mail message telling customers about a new product. The message is sent by a secretary, but the Reply-To: field lists the head of the sales department, who can answer questions and take orders. This field is also useful when the sender has two e-mail accounts and wants the reply to go to the other one.

The RFC 822 document explicitly says that users are allowed to invent new headers for their own private use, provided that these headers start with the string X-. It is guaranteed that no future headers will use names starting with X-, to avoid conflicts between official and private headers. Sometimes wiseguy undergraduates make up fields like X-Fruit-of-the-Day: or X-Disease-of-the-Week:, which are legal, although not always illuminating.

After the headers comes the message body. Users can put whatever they want here. Some people terminate their messages with elaborate signatures, including simple ASCII cartoons, quotations from greater and lesser authorities, political statements, and disclaimers of all kinds (e.g., The XYZ Corporation is not responsible for my opinions; in fact, it cannot even comprehend them).

MIME - The Multipurpose Internet Mail Extensions:

In the early days of the ARPANET, e-mail consisted exclusively of text messages written in English and expressed in ASCII. For this environment, RFC 822 did the job completely: it specified the headers but left the content entirely up to the users. Nowadays, on the worldwide Internet, this approach is no longer adequate. The problems include sending and receiving

- Messages in languages with accents (e.g., French and German).
- Messages in non-Latin alphabets (e.g., Hebrew and Russian).
- Messages in languages without alphabets (e.g., Chinese and Japanese).
- Messages not containing text at all (e.g., audio or images).

A solution was proposed in RFC 1341 and updated in RFCs 2045–2049. This solution, called MIME (Multipurpose Internet Mail Extensions) is now

widely used. We will now describe it. For additional information about MIME, see the RFCs.

The basic idea of MIME is to continue to use the RFC 822 format, but to add structure to the message body and define encoding rules for non-ASCII messages. By not deviating from RFC 822, MIME messages can be sent using the existing mail programs and protocols. All that has to be changed are the sending and receiving programs, which users can do for themselves.

MIME defines five new message headers, a The first of these simply tells the user agent receiving the message that it is dealing with a MIME message, and which version of MIME it uses. Any message not containing a MIME-Version: header is assumed to be an English plaintext message and is processed as such.

Message Transfer

The message transfer system is concerned with relaying messages from the originator to the recipient. The simplest way to do this is to establish a transport connection from the source machine to the destination machine and then just transfer the message. After examining how this is normally done, we will examine some situations in which this does not work and what can be done about them.

SMTP - The Simple Mail Transfer Protocol

Within the Internet, e-mail is delivered by having the source machine establish a TCP connection to port 25 of the destination machine. Listening to this port is an e-mail daemon that speaks **SMTP** (**Simple Mail Transfer Protocol**). This daemon accepts incoming connections and copies messages from them into the appropriate mailboxes. If a message cannot be delivered, an error report containing the first part of the undeliverable message is returned to the sender.

SMTP is a simple ASCII protocol. After establishing the TCP connection to port 25, the sending machine, operating as the client, waits for the receiving machine, operating as the server, to talk first. The server starts by sending a line of text giving its identity and telling whether it is prepared to receive mail. If it is not, the client releases the connection and tries again later.

If the server is willing to accept e-mail, the client announces whom the e-mail is coming from and whom it is going to. If such a recipient exists at the destination, the server gives the client the go-ahead to send the message. Then the client sends the message and the server acknowledges it. No checksums are needed because TCP provides a reliable byte stream. If there is more e-mail, that is now sent. When all the e-mail has been exchanged in both directions, the connection is released. including the numerical codes used by SMTP, The lines sent by the client are marked C:. Those sent by the server are marked S:.

Final Delivery

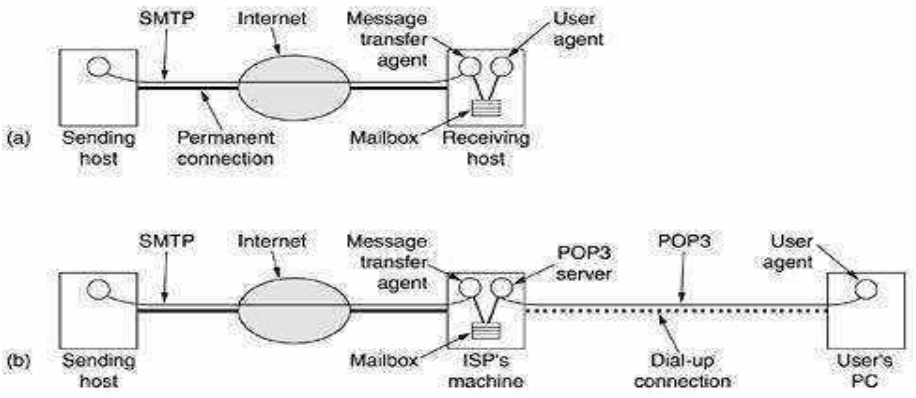
Up until now, we have assumed that all users work on machines that are capable of sending and receiving e-mail. As we saw, e-mail is delivered by having the sender establish a TCP connection to the receiver and then ship the e-mail over it. This model worked fine for decades when all ARPANET (and later Internet) hosts were, in fact, on-line all the time to accept TCP connections.

However, with the advent of people who access the Internet by calling their ISP over a modem, it breaks down. The problem is this: what happens when Elinor wants to send Carolyn e-mail and Carolyn is not currently on-line? Elinor cannot establish a TCP connection to Carolyn and thus cannot run the SMTP protocol.

One solution is to have a message transfer agent on an ISP machine accept e-mail for its customers and store it in their mailboxes on an ISP machine. Since this agent can be on-line all the time, e-mail can be sent to it 24 hours a day.

POP3

Unfortunately, this solution creates another problem: how does the user get the e-mail from the ISP's message transfer agent? The solution to this problem is to create another protocol that allows user transfer agents (on client PCs) to contact the message transfer agent (on the ISP's machine) and allow e-mail to be copied from the ISP to the user. One such protocol is **POP3** (**Post Office Protocol Version 3**), which is described in RFC 1939.



The situation that used to hold (both sender and receiver having a permanent connection to the Internet) .

Figure 7-15. (a) Sending and reading mail when the receiver has a permanent Internet connection and the user agent runs on the same machine as the message transfer agent. (b) Reading e-mail when the receiver has a dial-up connection to an ISP.

POP3 begins when the user starts the mail reader. The mail reader calls up the ISP (unless there is already a connection) and establishes a TCP connection with the message transfer agent at port 110. Once the connection has been established, the POP3 protocol goes through three states in sequence:

- Authorization.
- Transactions.
- Update.

The authorization state deals with having the user log in. The transaction state deals with the user collecting the e-mails and marking them for deletion from the mailbox. The update state actually causes the e-mails to be deleted.

IMAP

For a user with one e-mail account at one ISP that is always accessed from one PC, POP3 works fine and is widely used due to its simplicity and robustness. However, it is a computer-industry truism that as soon as something works well, somebody will start demanding more features (and getting more bugs). That happened with e-mail, too. For example, many people have a single e-mail account at work or school and want to access it from work, from their home PC, from their laptop when on business trips, and from cybercafes when on so-called

vacation. While POP3 allows this, since it normally downloads all stored messages at each contact, the result is that the user's e-mail quickly gets spread over multiple machines, more or less at random, some of them not even the user's.

This disadvantage gave rise to an alternative final delivery protocol, **IMAP** (**InternetMessage Access Protocol**), which is defined in RFC 2060. Unlike POP3, which basicallyassumes that the user will clear out the mailbox on every contact and work off-line after that, IMAP assumes that all the e-mail will remain on the server indefinitely in multiple mailboxes. IMAP provides extensive mechanisms for reading messages or even parts of messages, a feature useful when using a slow modem to read the text part of a multipart message with large audio and video attachments. Since the working assumption is that messages will not be transferred to the user's computer for permanent storage, IMAP provides mechanisms for creating, destroying, and manipulating multiple mailboxes on the server. In this way a user can maintain a mailbox for each correspondent and move messages there from the inbox after they have been read.

MAP has many features, such as the ability to address mail not by arrival number as is done, but by using attributes (e.g., give me the first message from Bobbie). Unlike POP3, IMAP can also accept outgoing e-mail for shipment to the destination as well as deliver incoming e-mail.

Delivery Features

Independently of whether POP3 or IMAP is used, many systems provide hooks for additional processing of incoming e-mail. An especially valuable feature for many e-mail users is the ability to set up **filters**. These are rules that are checked when e-mail comes in or when the user agent is started. Each rule specifies a condition and an action. For example, a rule could say that any message received from the boss goes to mailbox number 1, any message from a select group of friends goes to mailbox number 2, and any message containing certain objectionable words in the Subject line is discarded without comment.

Some ISPs provide a filter that automatically categorizes incoming e-mail as either important or spam (junk e-mail) and stores each message in the corresponding mailbox. Such filters typically work by first checking to see if the source is a known spammer. Then they usually examine the subject line. If hundreds of users have just received a message with the same subject line, it is probably spam. Other techniques are also used for spam detection.

Another delivery feature often provided is the ability to (temporarily) forward incoming e-mail to a different address. This address can even be a computer operated by a commercial paging service, which then pages the user by radio or satellite, displaying the Subject: line on his pager.

Webmail

One final topic worth mentioning is Webmail. Some Web sites, for example, Hotmail and Yahoo, provide e-mail service to anyone who wants it. They work as follows. They have normal message transfer agents listening to port 25 for incoming SMTP connections. To contact, say, Hotmail, you have to acquire their DNS MX record, for example, by typing host –a –v hotmail.com on a UNIX system. Suppose that the mail server is called mx10.hotmail.com, then by typing telnet mx10.hotmail.com 25. You can establish a TCP connection over which SMTP commands can be sent in the usual way. So far, nothing unusual, except that these big servers are often busy, so it may take several attempts to get a TCP connection accepted.

The interesting part is how e-mail is delivered. Basically, when the user goes to the e-mail Web page, a form is presented in which the user is asked for a login name and password. When the user clicks on Sign In, the login name and password are sent to the server, which then validates them. The Web page is then sent to the browser for display.Many of the items on the page are clickable, so messages can be read, deleted, and so on.

NETWORK SECURITY

Cryptography

Cryptography comes from the Greek words for "secret writing." It has a long and colorful history going back thousands of years. In this section we will just sketch some of the highlights, as background information for what follows. For a complete history of cryptography, Kahn's (1995) book is recommended reading. For a comprehensive treatment of the current state-of-the-art in security and cryptographic algorithms, protocols, and applications, see (Kaufman et al., 2002). For a more mathematical approach, see (Stinson, 2002). For a less mathematical approach, see (Burnett and Paine, 2001).

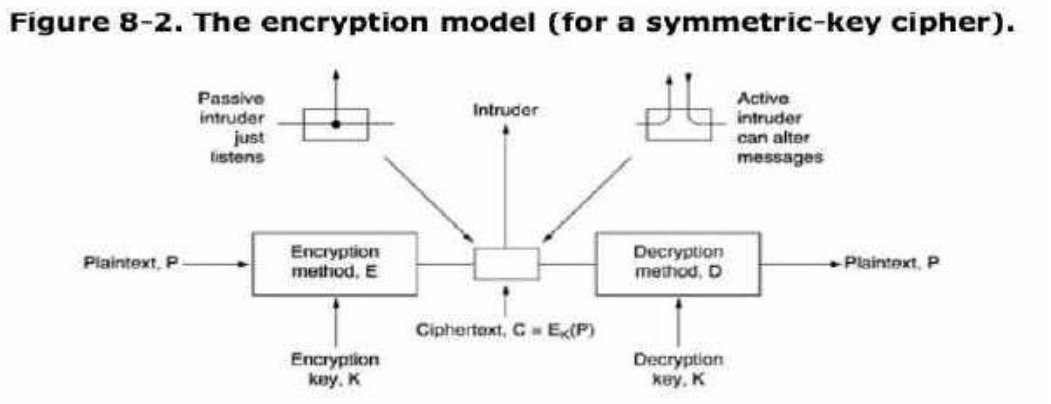
Professionals make a distinction between ciphers and codes. A **cipher** is a character-for-character or bit-for-bit transformation, without regard to the

linguistic structure of the message. In contrast, a **code** replaces one word with another word or symbol. Codes are not used any more, although they have a glorious history. The most successful code ever devised was used by the U.S. armed forces during World War II in the Pacific. They simply had Navajo Indians talking to each other using specific Navajo words for military terms, for example chay-dagahi-nail-tsaidi(literally: tortoise killer) for antitank weapon. The Navajo language is highlytonal, exceedingly complex, and has no written form. And not a single person in Japan knew anything about it.

Introduction to Cryptography

Historically, four groups of people have used and contributed to the art of cryptography: the military, the diplomatic corps, diarists, and lovers. Of these, the military has had the most important role and has shaped the field over the centuries. Within military organizations, the messages to be encrypted have traditionally been given to poorly-paid, low-level code clerks for encryption and transmission. The sheer volume of messages prevented this work from being done by a few elite specialists.

Figure 8-2. The encryption model (for a symmetric-key cipher).



Until the advent of computers, one of the main constraints on cryptography had been the ability of the code clerk to perform the necessary transformations, often on a battlefield with little equipment. An additional constraint has been the difficulty in switching over quickly from one cryptographic method to another one, since this entails retraining a large number of people. However, the danger of a code clerk being captured by the enemy has made it essential to be able to change the cryptographic method instantly if need be.

The messages to be encrypted, known as the **plaintext**, are transformed by a function that is parameterized by a **key**. The output of the encryption process, known as the **ciphertext**, is then transmitted, often by messenger or radio. We

assume that the enemy, or **intruder**, hears and accurately copies down the complete ciphertext. However, unlike the intended recipient, he does not know what the decryption key is and so cannot decrypt the ciphertext easily. Sometimes the intruder can not only listen to the communication channel (passive intruder) but can also record messages and play them back later, inject his own messages, or modify legitimate messages before they get to the receiver (active intruder). The art of breaking ciphers, called **cryptanalysis**, and the art devising them (cryptography) is collectively known as **cryptology**.

It will often be useful to have a notation for relating plaintext, ciphertext, and keys. We will use $C = E_K(P)$ to mean that the encryption of the plaintext P using key K gives the ciphertextC. Similarly, $P = D_K(C)$ represents the decryption of C to get the plaintext again. It then followsthat This notation suggests that E and D are just mathematical functions, which they are. The only tricky part is that both are functions of two parameters, and we have written one of the parameters (the key) as a subscript, rather than as an argument, to distinguish it from the message.

A fundamental rule of cryptography is that one must assume that the cryptanalyst knows the methods used for encryption and decryption. In other words, the cryptanalyst knows how the encryption method, E, and decryption, D,ofFig. 8-2 work in detail. The amount of effort necessary to invent, test, and install a new algorithm every time the old method is compromised (or thought to be compromised) has always made it impractical to keep the encryption algorithm secret. Thinking it is secret when it is not does more harm than good.

From the cryptanalyst's point of view, the cryptanalysis problem has three principal variations. When he has a quantity of ciphertext and no plaintext, he is confronted with the **ciphertext-only** problem. The cryptograms that appear in the puzzle section of newspapers pose this kindof problem. When the cryptanalyst has some matched ciphertext and plaintext, the problem is called the **known plaintext** problem. Finally, when the cryptanalyst has the ability to encrypt pieces of plaintext of his own choosing, we have the **chosen plaintext** problem. Newspaper cryptograms could be broken trivially if the cryptanalyst were allowed to ask such questions as: What is the encryption of ABCDEFGHIJKL?

Substitution Ciphers

In a **substitution cipher** each letter or group of letters is replaced by another letter or group of letters to disguise it. One of the oldest known ciphers is the **Caesar cipher**, attributed to Julius Caesar. In this method, a becomes D, b becomes E, c becomes F, ..., and z becomes C. For example, attack becomes DWWDFN. In examples, plaintext will be given in lower case letters, and cipher text in upper case letters.

A slight generalization of the Caesar cipher allows the cipher text alphabet to be shifted by k letters, instead of always 3. In this case k becomes a key to the general method of circularly shifted alphabets. The Caesar cipher may have fooled Pompey, but it has not fooled anyone since.

The next improvement is to have each of the symbols in the plaintext, say, the 26 letters for simplicity, and map onto some other letter.

For example,
Plaintext: a b c d e f g h i j k l m n o p q r s t u v w x y z
Cipher text : W E R T Y U I O P A S D F G H J K L Z X C V B N M

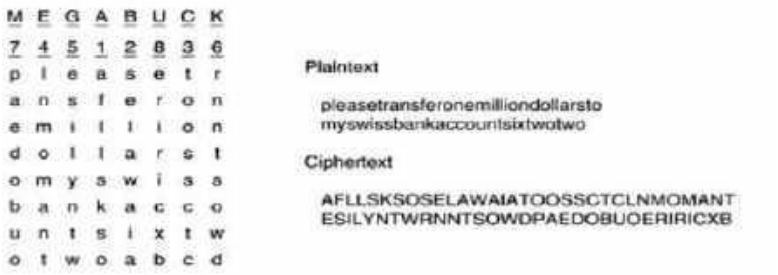
The general system of symbol-for-symbol substitution is called a **mono alphabetic substitution**, with the key being the 26-letter string corresponding to the full alphabet. For the key above, the plaintext attack would be transformed into the cipher text QZZQEA.

At first glance this might appear to be a safe system because although the cryptanalyst knows the general system (letter-for-letter substitution), he does not know which of the 26! $\approx 4 \times 10^{26}$ possible keys is in use. In contrast with the Caesar cipher, trying all of them is not a promising approach. Even at 1 nsec per solution, a computer would take 10^{10} years to try all the keys.

Transposition Ciphers

Substitution ciphers preserve the order of the plaintext symbols but disguise them. **Transposition ciphers**, in contrast, reorder the letters but do not disguise them. The cipher is keyed by a word or phrase not containing any repeated letters. In this example, MEGABUCK is the key.

Figure 8-3. A transposition cipher.



The purpose of the key is to number the columns, column 1 being under the key letter closest to the start of the alphabet, and so on. The plaintext is written horizontally, in rows, padded to fill the matrix if need be. The ciphertext is read out by columns, starting with the column whose key letter is the lowest.

To break a transposition cipher, the cryptanalyst must first be aware that he is dealing with a transposition cipher. By looking at the frequency of E, T, A, O, I, N, etc., it is easy to see if they fit the normal pattern for plaintext. If so, the cipher is clearly a transposition cipher, because in such a cipher every letter represents itself, keeping the frequency distribution intact.

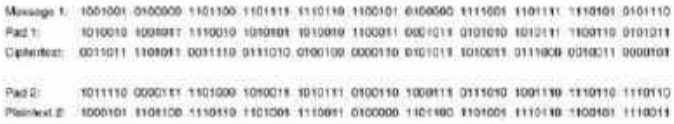
The next step is to make a guess at the number of columns. In many cases a probable word or phrase may be guessed at from the context. For example, suppose that our cryptanalyst suspects that the plaintext phrase million dollars occurs somewhere in the message. Observe that diagrams MO, IL, LL, LA, IR and OS occur in the cipher text as a result of this phrase wrapping around. The cipher text letter O follows the cipher text letter M (i.e., they are vertically adjacent in column 4) because they are separated in the probable phrase by a distance equal to the key length. If a key of length seven had been used, the diagrams MD, IO, LL, LL, IA, OR, and NS would have occurred instead. In fact, for each key length, a different set of diagrams is produced in the cipher text. By hunting for the various possibilities, the cryptanalyst can often easily determine the key length.

One-Time Pads

Constructing an unbreakable cipher is actually quite easy; the technique has been known for decades. First choose a random bit string as the key. Then convert the plaintext into a bit string, for example by using its ASCII representation. Finally, compute the XOR (exclusive OR) of these two strings,

bit by bit. The resulting cipher text cannot be broken, because in a sufficiently large sample of cipher text, each letter will occur equally often, as will every diagram, every trigram, and so on. This method, known as the **one-time pad**, is immune to all present and future attacks no matter how much computational power the intruder has. The reason derives from information theory: there is simply no information in the message because all possible plaintexts of the given length are equally likely.

Figure 8-4. The use of a one-time pad for encryption and the possibility of getting any possible plaintext from the ciphertext by the use of some other pad.



An example of how one-time pads are used is given in Fig. 8-4. First, message 1, "I love you." is converted to 7-bit ASCII. Then a one-time pad, pad 1, is chosen and XORed with the message to get the ciphertext. A cryptanalyst could try all possible one-time pads to see what plaintext came out for each one. For example, the one-time pad listed as pad 2 in the figure could be tried, resulting in plaintext 2, "Elvis lives", which may or may not be plausible (a subject beyond the scope of this book). In fact, for every 11-character ASCII plaintext, there is a one-time pad that generates it. That is what we mean by saying there is no information in the cipher text: you can get any message of the correct length out of it.

One-time pads are great in theory but have a number of disadvantages in practice. To start with, the key cannot be memorized, so both sender and receiver must carry a written copy with them. If either one is subject to capture, written keys are clearly undesirable.

Quantum Cryptography

Interestingly, there may be a solution to the problem of how to transmit the one-time pad over the network, and it comes from a very unlikely source: quantum mechanics. This area is still experimental, but initial tests are promising. If it can be perfected and be made efficient, virtually all cryptography will eventually be done using one-time pads since they are provably secure. Below we will briefly explain how this method, **quantum cryptography**, works. In

particular, we will describe a protocol called **BB84** after its authors and publication year (Bennet and Brassard, 1984).

Two Fundamental Cryptographic Principles

Although we will study many different cryptographic systems in the pages ahead, two principles underlying all of them are important to understand.

Redundancy

The first principle is that all encrypted messages must contain some redundancy, that is, information not needed to understand the message. An example may make it clear why this is needed. Consider a mail-order company, The Couch Potato (TCP), with 60,000 products. Thinking they are being very efficient, TCP's programmers decide that ordering messages should consist of a 16-byte customer name followed by a 3-byte data field (1 byte for the quantity and 2 bytes for the product number). The last 3 bytes are to be encrypted using a very long key known only by the customer and TCP.

At first this might seem secure, and in a sense it is because passive intruders cannot decrypt the messages. Unfortunately, it also has a fatal flaw that renders it useless. Suppose that a recently-fired employee wants to punish TCP for firing her. Just before leaving, she takes the customer list with her. She works through the night writing a program to generate fictitious orders using real customer names. Since she does not have the list of keys, she just puts random numbers in the last 3 bytes, and sends hundreds of orders off to TCP.

When these messages arrive, TCP's computer uses the customer's name to locate the key and decrypt the message. Unfortunately for TCP, almost every 3-byte message is valid, so the computer begins printing out shipping instructions. While it might seem odd for a customer to order 837 sets of children's swings or 540 sandboxes, for all the computer knows, the customer might be planning to open a chain of franchised playgrounds. In this way an active intruder (the ex-employee) can cause a massive amount of trouble, even though she cannot understand the messages her computer is generating.

Freshness

The second cryptographic principle is that some measures must be taken to ensure that each message received can be verified as being fresh, that is, sent very recently. This measure is needed to prevent active intruders from playing back

old messages. If no such measures were taken, our ex-employee could tap TCP's phone line and just keep repeating previously sent valid messages. Restating this idea we get:

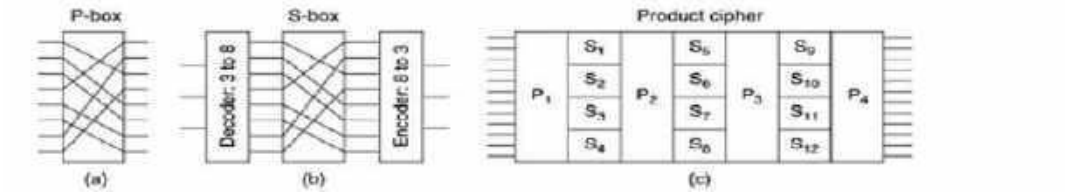
Cryptographic principle 2:

Some method is needed to foil replay attacks one such measure is including in every message a timestamp valid only for, say, 10 seconds. The receiver can then just keep messages around for 10 seconds, to compare newly arrived messages to previous ones to filter out duplicates. Messages older than 10 seconds can be thrown out, since any replays sent more than 10 seconds later will be rejected as too old. Measures other than timestamps will be discussed later.

8.2 Symmetric-Key Algorithms

Modern cryptography uses the same basic ideas as traditional cryptography (transposition and substitution) but its emphasis is different. Traditionally, cryptographers have used simple algorithms. Nowadays the reverse is true: the object is to make the encryption algorithm so complex and involute that even if the cryptanalyst acquires vast mounds of enciphered text of his own choosing, he will not be able to make any sense of it at all without the key.

8-6. Basic elements of product ciphers. (a) P-box. (b) S-box. (c) Product.



The first class of encryption algorithms we will study in this chapter are called **symmetric-key algorithms** because they used the same key for encryption and decryption. In particular, we will focus on **block ciphers**, which take an n-bit block of plaintext as input and transform it using the key into n-bit block of cipher text.

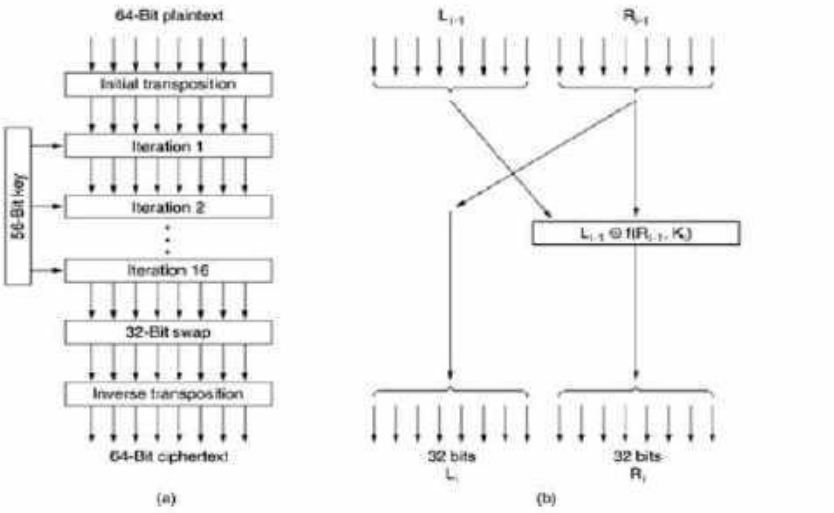
Cryptographic algorithms can be implemented in either hardware (for speed) or in software (for flexibility). Although most of our treatment concerns the algorithms and protocols, which are independent of the actual implementation, a few words about building cryptographic hardware may be of interest. Transpositions and substitutions can be implemented with simple

electrical circuit device, known as a **P-box** (P stands for permutation), used to effect a transposition on an 8-bit input. If the 8 bits are designated from top to bottom as 01234567, the output of this particular P-box is 36071245. By appropriate internal wiring, a P-box can be made to perform any transposition and do it at practically the speed of light since no computation is involved, just signal propagation. This design follows Kerckhoff's principle: the attacker knows that the general method is permuting the bits. What he does not know is which bit goes where, which is the key.

DES—The Data Encryption Standard

This cipher, **DES** (**Data Encryption Standard**), was widely adopted by the industry for use in security products. It is no longer secure in its original form, but in a modified form it is still useful. We will now explain how DES works.

Figure 8-7. The data encryption standard. (a) General outline. (b) Detail of one iteration. The circled + means exclusive OR.



Plaintext is encrypted in blocks of 64 bits, yielding 64 bits of cipher text. The algorithm, which is parameterized by a 56-bit key, has 19 distinct stages. The first stage is a key-independent transposition on the 64-bit plaintext. The last stage is the exact inverse of this transposition. The stage prior to the last one exchanges the leftmost 32 bits with the rightmost 32 bits. The remaining 16 stages are functionally identical but are parameterized by different functions of the key. The algorithm has been designed to allow decryption to be done with the same key as encryption, a property needed in any symmetric-key algorithm. The steps are just run in the reverse order.

The operation of each stage takes two 32-bit inputs and produces two 32-bit outputs. The left output is simply a copy of the right input. The right output is the bitwise XOR of the left input and a function of the right input and the key for this stage, K_i. All the complexity lies in this function.

The function consists of four steps, carried out in sequence. First, a 48-bit number, E, is constructed by expanding the 32-bit R_{i-1} according to a fixed transposition and duplication rule. Second, E and K_i are XORed together. This output is then partitioned into eight groups of 6 bits each, each of which is fed into a different S-box. Each of the 64 possible inputs to an S-box is mapped onto a 4-bit output. Finally, these 8 x 4 bits are passed through a P-box.

AES—The Advanced Encryption Standard

As DES began approaching the end of its useful life, even with triple DES, **NIST** (**National Institute of Standards and Technology**), the agency of the U.S. Dept. of Commerce charged with approving standards for the U.S. Federal Government, decided that the government needed a new cryptographic standard for unclassified use. NIST was keenly aware of all the controversy surrounding DES and well knew that if it just announced a new standard, everyone knowing anything about cryptography would automatically assume that NSA had built a back door into it so NSA could read everything encrypted with it. Under these conditions, probably no one would use the standard and it would most likely die a quiet death.

So NIST took a surprisingly different approach for a government bureaucracy: it sponsored a cryptographic bake-off (contest). In January 1997, researchers from all over the world were invited to submit proposals for a new standard, to be called **AES** (**Advanced Encryption Standard**).

The bake-off rules were:

- The algorithm must be a symmetric block cipher.
- The full design must be public.
- Key lengths of 128, 192, and 256 bits must be supported.
- Both software and hardware implementations must be possible.
- The algorithm must be public or licensed on nondiscriminatory terms.

Fifteen serious proposals were made, and public conferences were organized in which they were presented and attendees were actively encouraged to find flaws in all of them. In August 1998, NIST selected five finalists primarily on the basis of their security, efficiency, simplicity, flexibility, and memory requirements (important for embedded systems). More conferences were held and more pot-shots taken. A nonbinding vote was taken at the last conference. The finalists and their scores were as follows:

- Rijndael (from Joan Daemen and Vincent Rijmen, 86 votes).
- Serpent (from Ross Anderson, Eli Biham, and Lars Knudsen, 59 votes).
- Twofish (from a team headed by Bruce Schneier, 31 votes).
- RC6 (from RSA Laboratories, 23 votes).
- MARS (from IBM, 13 votes).

In October 2000, NIST announced that it, too, voted for Rijndael, and in November 2001 Rijndael became a U.S. Government standard published as Federal Information Processing Standard FIPS 197. Due to the extraordinary openness of the competition, the technical properties of Rijndael, and the fact that the winning team consisted of two young Belgian cryptographers (who are unlikely to have built in a back door just to please NSA), it is expected that Rijndael will become the world's dominant cryptographic standard for at least a decade. The name Rijndael, pronounced Rhine-doll (more or less), is derived from the last names of the authors: Rijmen + Daemen.

Rijndael supports key lengths and block sizes from 128 bits to 256 bits in steps of 32 bits. The key length and block length may be chosen independently. However, AES specifies that the block size must be 128 bits and the key length must be 128, 192, or 256 bits. It is doubtful that anyone will ever use 192-bit keys, so de facto, AES has two variants: a 128-bit block with 128-bit key and a 128-bit block with a 256-bit key.

In our treatment of the algorithm below, we will examine only the 128/128 case because this is likely to become the commercial norm. A 128-bit key gives a key space of $2^{128} \approx 3 \times 103^{8}$ keys. Even if NSA manages to build a machine with 1 billion parallel processors, each being able to evaluate one key per picosecond, it would take such a machine about 10^{10} years to search the key space.

By then the sun will have burned out, so the folks then present will have to read the results by candlelight.

Like DES, Rijndael uses substitution and permutations, and it also uses multiple rounds. The number of rounds depends on the key size and block size, being 10 for 128-bit keys with 128-bit blocks and moving up to 14 for the largest key or the largest block. However, unlike DES, all operations involve entire bytes, to allow for efficient implementations in both hardware and software.

The function rijndael has three parameters. They are: plaintext, an array of 16 bytes containing the input data, ciphertext, an array of 16 bytes where the enciphered output will be returned, and key, the 16-byte key. During the calculation, the current state of the data is maintained in a byte array, state, whose size is NROWS x NCOLS. For 128-bit blocks, this array is 4 x 4 bytes. With 16 bytes, the full 128-bit data block can be stored.

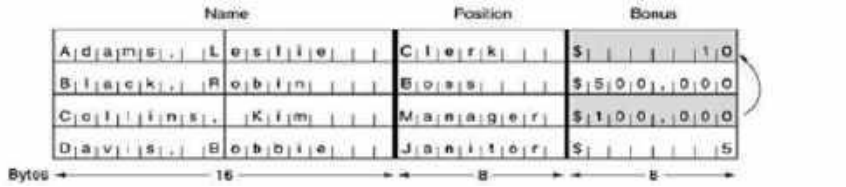
The state array is initialized to the plaintext and modified by every step in the computation. In some steps, byte-for-byte substitution is performed. In others, the bytes are permuted within the array. Other transformations are also used. At the end, the contents of the state are returned as the ciphertext.

The code starts out by expanding the key into 11 arrays of the same size as the state. They are stored in rk, which is an array of structs, each containing a state array. One of these will be used at the start of the calculation and the other 10 will be used during the 10 rounds, one per round. The calculation of the round keys from the encryption key is too complicated for us to get into here. Suffice it to say that the round keys are produced by repeated rotation and XORing of various groups of key bits. For all the details, see (Daemen and Rijmen, 2002).

Cipher Modes

Despite all this complexity, AES (or DES or any block cipher for that matter) is basically a monoalphabetic substitution cipher using big characters (128-bit characters for AES and 64-bit characters for DES). Whenever the same plaintext block goes in the front end, the same ciphertext block comes out the back end. If you encrypt the plaintext abcdefgh 100 times with the same DES key, you get the same ciphertext 100 times. An intruder can exploit this property to help subvert the cipher.

Figure 8-11. The plaintext of a file encrypted as 16 DES blocks.



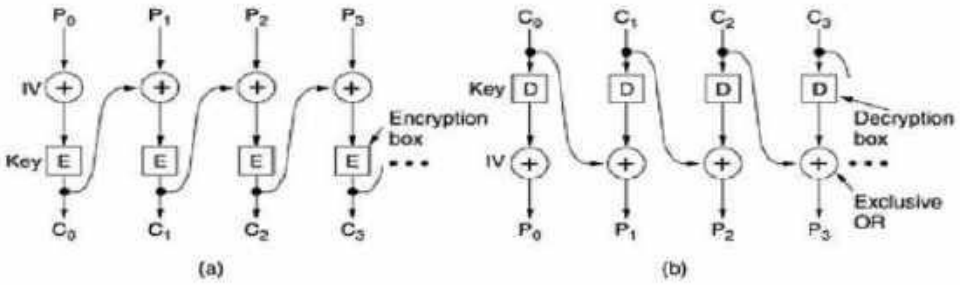
Electronic Code Book Mode

To see how this mono alphabetic substitution cipher property can be used to partially defeat the cipher, we will use (triple) DES because it is easier to depict 64-bit blocks than 128-bit blocks, but AES has exactly the same problem. The straightforward way to use DES to encrypt a long piece of plaintext is to break it up into consecutive 8-byte (64-bit) blocks and encrypt them one after another with the same key. The last piece of plaintext is padded out to 64 bits, if need be. This technique is known as **ECB mode** (**Electronic Code Book mode**) in analogy with old-fashioned code books where each plaintext word was listed, followed by its cipher text (usually a five-digit decimal number).

We have the start of a computer file listing the annual bonuses a company has decided to award to its employees. This file consists of consecutive 32-byte records, one per employee, in the format shown: 16 bytes for the name, 8 bytes for the position, and 8 bytes for the bonus. Each of the sixteen 8-byte blocks (numbered from 0 to 15) is encrypted by (triple) DES.

Cipher Block Chaining Mode

To thwart this type of attack, all block ciphers can be chained in various ways so that replacing a block the way Leslie did will cause the plaintext decrypted starting at the replaced block to be garbage. One way of chaining is **cipher block chaining**. In this method, shown in Fig. 8-12, each plaintext block is XORed with the previous cipher text block before being encrypted. Consequently, the same plaintext block no longer maps onto the same cipher text block, and the encryption is no longer a big monoalphabetic substitution cipher. The first block is XORed with a randomly chosen **IV** (**Initialization Vector**), which is transmitted (in plaintext) along with the ciphertext.

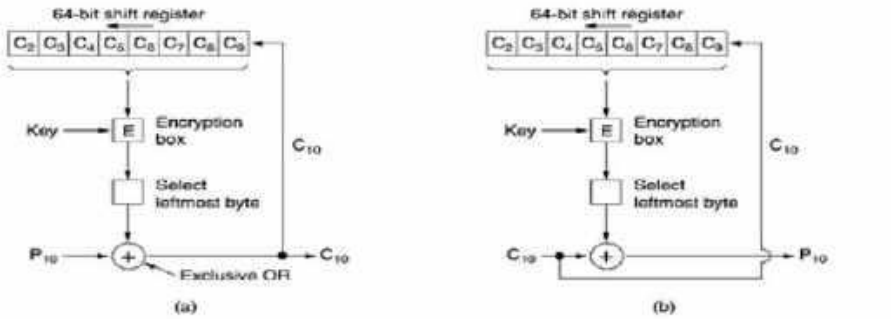


Cipher block chaining. (a) Encryption. (b) Decryption

We can see how cipher block chaining mode works by examining the example of We start out by computing $C_0 = E(P_0 \text{ XOR } IV)$. Then we compute $C_1 = E(P_1 \text{ XOR } C_0)$, and so on. Decryption also uses XOR to reverse the process, with $P_0 = IV \text{ XOR } D(C_0)$, and so on. Note that the encryption of block iis a function of all the plaintext in blocks 0 through i - 1, so the same plaintext generates different ciphertext depending on where it occurs. A transformation of the type Leslie made will result in nonsense for two blocks starting at Leslie's bonus field. To an astute security officer, this peculiarity might suggest where to start the ensuing investigation.

Cipher Feedback Mode

However, cipher block chaining has the disadvantage of requiring an entire 64-bit block to arrive before decryption can begin. For use with interactive terminals, where people can type lines shorter than eight characters and then stop, waiting for a response, this mode is unsuitable. For byte-by-byte encryption, **cipher feedback mode**, using (triple) DES is used, for AES the idea is exactly the same, only a 128-bit shift register is used. In this figure, the state of the encryption machine is shown after bytes 0 through 9 have been encrypted and sent. When plaintext byte 10 arrives, , the DES algorithm operates on the 64-bit shift register to generate a 64-bit ciphertext. The leftmost byte of that ciphertext is extracted and XORed with P_{10}. That byte is transmitted on the transmission line. In addition, the shift register is shifted left 8 bits, causing C_2 to fall off the left end, and C_{10} is inserted in the position just vacated at the right end by C_9. Note that the contents of the shift register depend on the entire previous history of the plaintext, so a pattern that repeats multiple times in the plaintext will be encrypted differently each time in the ciphertext. As with cipher block chaining, an initialization vector is needed to start the ball rolling.



Cipher feedback mode. (a) Encryption. (b) Decryption.

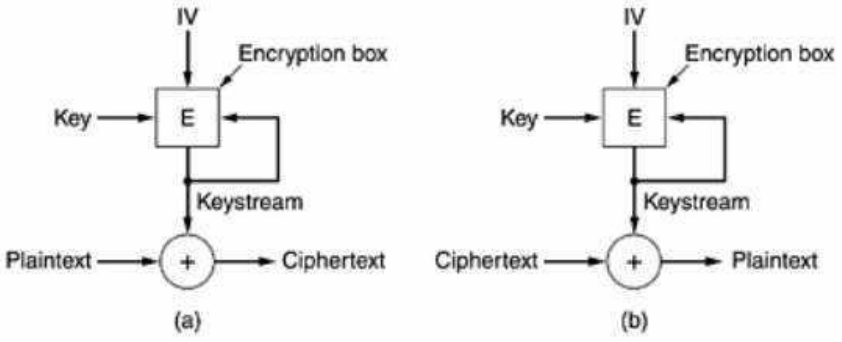
Decryption with cipher feedback mode just does the same thing as encryption. In particular, the content of the shift register is encrypted, not decrypted, so the selected byte that is XORed with C_{10} to get P_{10} is the same one that was XORed with P_{10} to generate C_{10} in the first place. As long as the two shift registers remain identical, decryption works correctly.

Stream Cipher Mode

It works by encrypting an initialization vector, using a key to get an output block. The output block is then encrypted, using the key to get a second output block. This block is then encrypted to get a third block, and so on. The (arbitrarily large) sequence of output blocks, called the **keystream**, is treated like a one-time pad and XORed with the plaintext to get the cipher text. Note that the IV is used only on the first step. After that, the output is encrypted. Also note that the keystream is independent of the data, so it can be computed in advance, if need be, and is completely insensitive to transmission errors.

Figure 8-14. A stream cipher. (a) Encryption. (b) Decryption.

8-14. A stream cipher. (a) Encryption. (b) Decryption.



Decryption occurs by generating the same key stream at the receiving side. Since the key stream depends only on the IV and the key, it is not affected by transmission errors in the cipher text. Thus, a 1-bit error in the transmitted cipher text generates only a 1-bit error in the decrypted plaintext.

It is essential never to use the same (key, IV) pair twice with a stream cipher because doing so will generate the same key stream each time. Using the same key stream twice exposes the cipher text to a **key stream reuse attack**. Imagine that the plaintext block, P_0, is encrypted with the key stream to get P_0 XOR K_0. Later, a second plaintext block, Q_0, is encrypted with the same key stream to get Q_0 XOR K_0. An intruder who captures both of these ciphertext blocks can simply XOR them together to get P_0 XOR Q_0, which eliminates the key. The intruder now has the XOR of the two plaintext blocks. If one of them is known or can be guessed, the other can also be found. In any event, the XOR of two plaintext streams can be attacked by using statistical properties of the message. For example, for English text, the most common character in the stream will probably be the XOR of two spaces, followed by the XOR of space and the letter "e", etc. In short, equipped with the XOR of two plaintexts, the cryptanalyst has an excellent chance of deducing both of them.

Public-Key Algorithms

In their proposal, the (keyed) encryption algorithm, E, and the (keyed) decryption algorithm, D, had to meet three requirements.

These requirements can be stated simply as follows:

- D(E(P)) = P.
- It is exceedingly difficult to deduce D from E.
- E cannot be broken by a chosen plaintext attack.

The first requirement says that if we apply D to an encrypted message, E(P), we get the original plaintext message, P, back. Without this property, the legitimate receiver could not decrypt the cipher text. The second requirement speaks for itself. The third requirement is needed because, as we shall see in a moment, intruders may experiment with the algorithm to their hearts' content. Under these conditions, there is no reason that the encryption key cannot be made public.

The method works like this. A person, say, Alice, wanting to receive secret messages, first devises two algorithms meeting the above requirements. The encryption algorithm and Alice's key are then made public, hence the name **public-key cryptography**. Alice might put her public key on her home page on the Web, for example. We will use the notation E_A to mean the encryption algorithm parameterized by Alice's public key. Similarly, the (secret) decryption algorithm parameterized by Alice's private key is D_A. Bob does the same thing, publicizing E_B but keeping D_B secret.

RSA

The only catch is that we need to find algorithms that indeed satisfy all three requirements. Due to the potential advantages of public-key cryptography, many researchers are hard at work, and some algorithms have already been published. One good method was discovered by a group at M.I.T. (Rivest et al., 1978). It is known by the initials of the three discoverers (Rivest, Shamir, Adleman): **RSA**. It has survived all attempts to break it for more than a quarter of a century and is considered very strong. Much practical security is based on it. Its major disadvantage is that it requires keys of at least 1024 bits for good security (versus 128 bits for symmetric-key algorithms), which makes it quite slow.

An example of the RSA algorithm

The RSA method is based on some principles from number theory. We will now summarize how to use the method; for details, consult the paper.

- Choose two large primes, p and q (typically 1024 bits).
- Compute n = p x q and z = (p - 1) x (q - 1).
- Choose a number relatively prime to z and call it d.
- Find e such that e x d = 1 mod z.

With these parameters computed in advance, we are ready to begin encryption. Divide the plaintext (regarded as a bit string) into blocks, so that each plaintext message, P, falls in the interval $0 \leq P < n$. Do that by grouping the plaintext into blocks of k bits, where k is the largest integer for which $2^k < n$ is true.

To encrypt a message, P, compute $C = P^e \pmod n$. To decrypt C, compute $P = C^d \pmod n$. It can be proven that for all P in the specified range, the encryption and decryption functions are inverses. To perform the encryption, you need e and n. To perform the decryption, you need d and n. Therefore, the public key consists of the pair (e, n), and the private key consists of (d, n).

The security of the method is based on the difficulty of factoring large numbers. If the cryptanalyst could factor the (publicly known) n, he could then find p and q, and from these z. Equipped with knowledge of z and e, d can be found using Euclid's algorithm. Fortunately, mathematicians have been trying to factor large numbers for at least 300 years, and the accumulated evidence suggests that it is an exceedingly difficult problem.

According to Rivest and colleagues, factoring a 500-digit number requires 10^{25} years using brute force. In both cases, they assume the best known algorithm and a computer with a 1-µsec instruction time. Even if computers continue to get faster by an order of magnitude per decade, it will be centuries before factoring a 500-digit number becomes feasible, at which time our descendants can simply choose p and q still larger.

A trivial pedagogical example of how the RSA algorithm works for this example we have chosen p = 3 and q = 11, giving n = 33 and z = 20. A suitable value for d is d = 7, since 7 and 20 have no common factors. With these choices, e can be found by solving the equation $7e = 1 \pmod{20}$, which yields e = 3. The cipher text, C, for a plaintext message, P, is given by $C = P^3 \pmod{33}$. The cipher text is decrypted by the receiver by making use of the rule $P = C^7 \pmod{33}$.

Because the primes chosen for this example are so small, P must be less than 33, so each plaintext block can contain only a single character. The result is a monoalphabetic substitution cipher, not very impressive. If instead we had chosen p and q $\approx 2^{512}$, we would have n $\approx 2^{1024}$, so each block could be up to

1024 bits or 128 eight-bit characters, versus 8 characters for DES and 16 characters for AES.

Digital Signatures

The authenticity of many legal, financial, and other documents is determined by the presence or absence of an authorized handwritten signature. And photocopies do not count. For computerized message systems to replace the physical transport of paper and ink documents, a method must be found to allow documents to be signed in an unforgeable way.

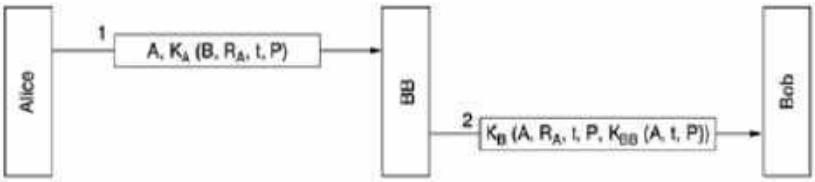
The problem of devising a replacement for handwritten signatures is a difficult one. Basically, what is needed is a system by which one party can send a signed message to another party in such a way that the following conditions hold:

- The receiver can verify the claimed identity of the sender.
- The sender cannot later repudiate the contents of the message.
- The receiver cannot possibly have concocted the message himself.

The first requirement is needed, for example, in financial systems. When a customer's computer orders a bank's computer to buy a ton of gold, the bank's computer needs to be able to make sure that the computer giving the order really belongs to the company whose account is to be debited. In other words, the bank has to authenticate the customer (and the customer has to authenticate the bank).

The second requirement is needed to protect the bank against fraud. Suppose that the bank buys the ton of gold, and immediately thereafter the price of gold drops sharply. A dishonest customer might sue the bank, claiming that he never issued any order to buy gold. When the bank produces the message in court, the customer denies having sent it. The property that no party to a contract can later deny having signed it is called **non repudiation**. The digital signature schemes that we will now study help provide it.

Figure 8-18. Digital signatures with Big Brother.



The third requirement is needed to protect the customer in the event that the price of gold shoots up and the bank tries to construct a signed message in which the customer asked for one bar of gold instead of one ton. In this fraud scenario, the bank just keeps the rest of the gold for itself.

Symmetric-Key Signatures

One approach to digital signatures is to have a central authority that knows everything and whom everyone trusts, say Big Brother (BB). Each user then chooses a secret key and carries it by hand to BB's office. Thus, only Alice and BB know Alice's secret key, K_A, and so on.

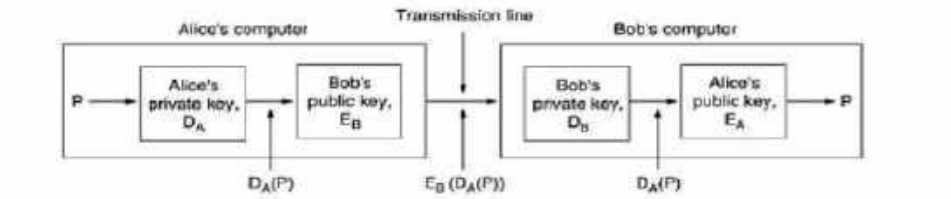
When Alice wants to send a signed plaintext message, P, to her banker, Bob, she generates $K_A(B, R_A, t, P)$, where B is Bob's identity, $R_A$is a random number chosen by Alice, t is a time stamp to ensure freshness, and $K_A(B, R_A, t, P)$ is the message encrypted with her key, K_A. BB sees that the message is from Alice, decrypts it, and what happens if Alice later denies sending the message? Step 1 is that everyone sues everyone (at least, in the United States). Finally, when the case comes to court and Alice vigorously denies sending Bob the disputed message, the judge will ask Bob how he can be sure that the disputed message came from Alice and not from Trudy. Bob first points out that BB will not accept a message from Alice unless it is encrypted with K_A, so there is no possibility of Trudy sending BB a false message from Alice without BB detecting it immediately.

Bob then dramatically produces Exhibit A: K_{BB} (A, t, P). Bob says that this is a message signed by BB which proves Alice sent P to Bob. The judge then asks BB (whom everyone trusts) to decrypt Exhibit A. When BB testifies that Bob is telling the truth, the judge decides in favor of Bob. Case dismissed.

Public-Key Signatures

A structural problem with using symmetric-key cryptography for digital signatures is that everyone has to agree to trust Big Brother. Furthermore, Big Brother gets to read all signed messages. The most logical candidates for running the Big Brother server are the government, the banks, the accountants, and the lawyers. Unfortunately, none of these organizations inspire total confidence in all citizens. Hence, it would be nice if signing documents did not require a trusted authority.

Figure 8-19. Digital signatures using public-key cryptography.



Fortunately, public-key cryptography can make an important contribution in this area. Let us assume that the public-key encryption and decryption algorithms have the property that E(D(P)) = P in addition, of course, to the usual property that D(E(P)) = P. (RSA has this property, so the assumption is not unreasonable.) Assuming that this is the case, Alice can send a signed plaintext message, P, to Bob by transmitting $E_B(D_A(P))$. Note carefully that Alice knows her own (private) key, D_A, as well as Bob's public key, E_B, so constructing this message is something Alice can do.

When Bob receives the message, he transforms it using his private key, as usual, yielding $D_A(P)$, He stores this text in a safe place and then applies $E_A$to get the original plaintext.

To see how the signature property works, suppose that Alice subsequently denies having sent the message P to Bob. When the case comes up in court, Bob can produce both P and $D_A(P)$. The judge can easily verify that Bob indeed has a valid message encrypted by D_A by simply applying E_A to it. Since Bob does not know what Alice's private key is, the only way Bob could have acquired a message encrypted by it is if Alice did indeed send it. While in jail for perjury and fraud, Alice will have plenty of time to devise interesting new public-key algorithms.

In principle, any public-key algorithm can be used for digital signatures. The de facto industry standard is the RSA algorithm. Many security products use it. However, in 1991, NIST proposed using a variant of the El Gamal public-key algorithm for their new **Digital Signature Standard** (**DSS**). El Gamal gets its security from the difficulty of computing discrete logarithms, rather than from the difficulty of factoring large numbers.

As usual when the government tries to dictate cryptographic standards, there was an uproar. DSS was criticized for being

- Too secret (NSA designed the protocol for using El Gamal).
- Too slow (10 to 40 times slower than RSA for checking signatures).
- Too new (El Gamal had not yet been thoroughly analyzed).
- Too insecure (fixed 512-bit key).

In a subsequent revision, the fourth point was rendered moot when keys up to 1024 bits were allowed. Nevertheless, the first two points remain valid.

Message Digests

One criticism of signature methods is that they often couple two distinct functions: authentication and secrecy. Often, authentication is needed but secrecy is not. Also, getting an export license is often easier if the system in question provides only authentication but not secrecy. Below we will describe an authentication scheme that does not require encrypting the entire message.

This scheme is based on the idea of a one-way hash function that takes an arbitrarily long piece of plaintext and from it computes a fixed-length bit string. This hash function, MD, often called a **message digest**, has four important properties:

- Given P, it is easy to compute MD (P).
- Given MD (P), it is effectively impossible to find P.
- Given P no one can find P' such that MD (P') = MD (P).
- A change to the input of even 1 bit produces a very different output.

To meet criterion 3, the hash should be at least 128 bits long, preferably more. To meet criterion 4, the hash must mangle the bits very thoroughly, not unlike the symmetric-key encryption algorithms we have seen.

Computing a message digest from a piece of plaintext is much faster than encrypting that plaintext with a public-key algorithm, so message digests can be used to speed up digital signature algorithms. Instead of signing P with K_{BB} (A, t, P), BB now computes the message digest by applying MD to P, yielding MD (P). BB then encloses K_{BB} (A, t, MD (P)) as the fifth item in the list encrypted with K_B that is sent to Bob, instead of K_{BB} (A, t, P).

Digital signatures using message digests.

If a dispute arises, Bob can produce both P and K_{BB} (A, t, MD (P)). After Big Brother has decrypted it for the judge, Bob has MD (P), which is guaranteed to be genuine, and the alleged P. However, since it is effectively impossible for Bob to find any other message that gives this hash, the judge will easily be convinced that Bob is telling the truth. Using message digests in this way saves both encryption time and message transport costs.

Message digests work in public-key cryptosystems, too, Here, Alice first computes the message digest of her plaintext. She then signs the message digest and sends both the signed digest and the plaintext to Bob. If Trudy replaces P underway, Bob will see this when he computes MD (P) himself.

MD5

A variety of message digest functions have been proposed. The most widely used ones are MD5 (Rivest, 1992) and SHA-1 (NIST, 1993). **MD5** is the fifth in a series of message digests designed by Ronald Rivest. It operates by mangling bits in a sufficiently complicated way that every output bit is affected by every input bit. Very briefly, it starts out by padding the message to a length of 448 bits (modulo 512). Then the original length of the message is appended as a 64-bit integer to give a total input whose length is a multiple of 512 bits. The last pre-computation step is initializing a 128-bit buffer to a fixed value.

Now the computation starts. Each round takes a 512-bit block of input and mixes it thoroughly with the 128-bit buffer. For good measure, a table constructed

from the sine function is also thrown in. The point of using a known function like the sine is not because it is more random than a random number generator, but to avoid any suspicion that the designer built in a clever back door through which only he can enter. Remember that IBM's refusal to disclose the principles behind the design of the S-boxes in DES led to a great deal of speculation about back doors. Rivest wanted to avoid this suspicion. Four rounds are performed per input block. This process continues until all the input blocks have been consumed. The contents of the 128-bit buffer form the message digest.

MD5 has been around for over a decade now, and many people have attacked it. Some vulnerabilities have been found, but certain internal steps prevent it from being broken. However, if the remaining barriers within MD5 fall, it may eventually fail. Nevertheless, at the time of this writing, it was still standing.

SHA-1

The other major message digest function is **SHA-1** (**Secure Hash Algorithm 1**), developed by NSA and blessed by NIST in FIPS 180-1. Like MD5, SHA-1 processes input data in 512-bit blocks, only unlike MD5, it generates a 160-bit message digest. A typical way for Alice to send a non-secret but signed message to Bob is illustrated in Fig. 8-21. Here her plaintext message is fed into the SHA-1 algorithm to get a 160-bit SHA-1 hash. Alice then signs the hash with her RSA private key and sends both the plaintext message and the signed hash to Bob.

8-21. Use of SHA-1 and RSA for signing nonsecret messages.

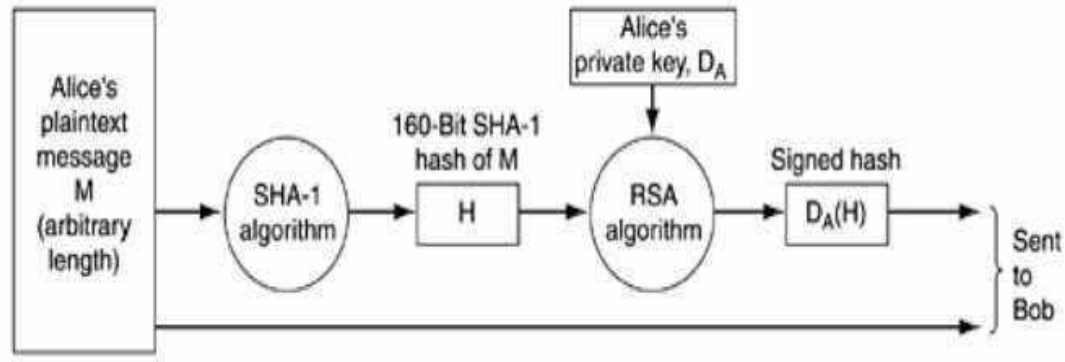
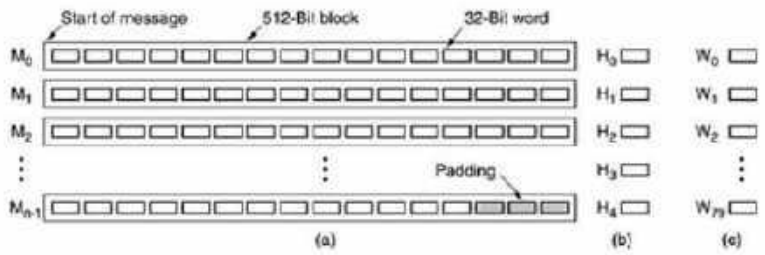


Figure 8-22. (a) A message padded out to a multiple of 512 bits. (b) The output variables. (c) The word array.



After receiving the message, Bob computes the SHA-1 hash himself and also applies Alice's public key to the signed hash to get the original hash, H. If the two agree, the message is considered valid. Since there is no way for Trudy to modify the (plaintext) message while its is in transit and produce a new one that hashes to H, Bob can easily detect any changes Trudy has made to the message. For messages whose integrity is important but whose contents are not secret. For a relatively small cost in computation, it guarantees that any modifications made to the plaintext message in transit can be detected with very high probability.

Now let us briefly see how SHA-1 works. It starts out by padding the message by adding a 1 bit to the end, followed by as many 0 bits as are needed to make the length a multiple of 512 bits. Then a 64-bit number containing the message length before padding is ORed into the low-order 64 bits. The message is shown with padding on the right because English text and figures go from left to right (i.e., the lower right is generally perceived as the end of the figure). With computers, this orientation corresponds to big-endian machines such as the SPARC, but SHA-1 always pads the end of the message, no matter which endian machine is used.

During the computation, SHA-1 maintains five 32-bit variables, H_0 through H_4, They are initialized to constants specified in the standard.

Each of the blocks M_0 through M_{n-1} is now processed in turn. For the current block, the 16 words are first copied into the start of an auxiliary 80-word array, W, Then the other 64 words in W are filled in using the formula

$$W_i = S^1(W_{i-3} \text{ XOR } W_{i-8} \text{ XOR } W_{i-14} \text{ XOR } W_{i-16}) \quad (16 \le i \le 79)$$

where$S^b(W)$ represents the left circular rotation of the 32-bit word, W, by b bits. Now five scratch variables, A through E are initialized from H_0 through H_4, respectively.

The actual calculation can be expressed in pseudo-C as

for (i = 0; i < 80; i++) {
temp = $S^5(A) + f_i (B, C, D) + E + W_i + K_i$;
E=D; D=C; C=S^{30}(B); B = A; A = temp;}

where the K_i constants are defined in the standard. The mixing functions f_i are defined as

$$f_i(B,C,D) = (B \text{ AND } C) \text{ OR } (\text{NOT } B \text{ AND } D) \quad (0 \le i \le 19)$$
$$f_i(B,C,D) = B \text{ XOR } C \text{ XOR } D \quad (20 \le i \le 39)$$
$$f_i(B,C,D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D) \quad (40 \le i \le 59)$$
$$f_i(B,C,D) = B \text{ XOR } C \text{ XOR } D \quad (60 \le i \le 79)$$

When all 80 iterations of the loop are completed, A through Eare added to H_0 through H_4, respectively.

Now that the first 512-bit block has been processed, the next one is started. The W array is reinitialized from the new block, but H is left as it was. When this block is finished, the next one is started, and so on, until all the 512-bit message blocks have been tossed into the soup. When the last block has been finished, the five 32-bit words in the H array are output as the 160-bit cryptographic hash. The complete C code for SHA-1 is given in RFC 3174.