BCA

THIRD YEAR

CORE PAPER - 7

COMPUTER NETWORKS

BHARATHIAR UNIVERSITY
SCHOOL OF DISTANCE EDUCATION
COIMBATORE - 641 046

CORE - 7 COMPUTER NETWORKS

SYLLABUS

Subject Description: This subject deals different Network concepts like Layers, Wireless Concepts, Transmission and Security.

Goal: Knowledge on Computer Networks and technologies like broadband and Bluetooth.

Objective: To inculcate knowledge on Networking concepts and echnologies like wireless, broadband and Bluetooth.

UNIT-I: Network Hardware: LAN – WAN – MAN – Wireless – Home Networks. **Network Software:** Protocol Hierarchies – Design Issues for the Layers – Connection-oriented and connectionless services – Service Primitives – The Relationship of services to Protocols. **Reference Models:** OSI Reference Model – TCP/IP reference Model – Comparison of OSI and TCP/IP -Critique of OSI and protocols – Critique of the TCP/IP Reference model.

UNIT-II: PHYSICAL LAYER - Guided Transmission Media: Magnetic Media – Twisted Pair – Coaxial Cable – Fiber Optics. **Wireless Transmission:** Electromagnetic Spectrum – Radio Transmission – Microwave Transmission – Infrared and Millimeter Waves – Light Waves. **Communication Satellites:** Geostationary, Medium-Earth Orbit, Low Earth-orbit Satellites – Satellites versus Fiber.

UNIT-III: DATA-LINK LAYER: Error Detection and correction – Elementary Data-link Protocols – Sliding Window Protocols. **MEDIUM-ACCESS CONTROL SUB LAYER:** Multiple Access Protocols – Ethernet – Wireless LANs - Broadband Wireless – Bluetooth.

UNIT-IV: NETWORK LAYER: Routing algorithms – Congestion Control Algorithms. TRANSPORT LAYER: Elements of Transport Protocols – Internet Transport Protocols: TCP.

UNIT-V: APPLICATION LAYER: DNS – E-mail. **NETWORK SECURITY:** Cryptography – Symmetric Key Algorithms – Public Key Algorithms – Digital Signatures.

TEXTBOOKS:

1. **COMPUTER NETWORKS – Andrew S. Tanenbaum, 4th edition, PHI.**

(UNIT-I:1.2-1.4 UNIT-II:2.2-2.4 UNIT-III:4.2-4.6 UNIT-IV:5.2,5.3,6.2,6.5 UNIT-V:7.1,7.2,8.1-8.4)

REFERENCE BOOKS:

1. **DATA COMMUNICATION AND NETWORKS – Achyut Godbole, 2007, TMH.**
2. **COMPUTER NETWORKS Protocols, Standards, and Interfaces – Uyless Black, 2nd ed, PHI.**

CONTENTS

UNIT - I

Lesson 1

Network Hardware

Contents

1.0 Aim

This chapter aims at introducing the various concepts about the network hardware which forms as the backbone of any computer networks. Physical mediums and logical topologies for connecting the networks are discussed here.

1.1 Introduction

The technical issues involved in network design are discussed in detail in this section. There are a lot of things on which computer networks depend on, but two dimensions stand out as important:

1. Transmission Technology
2. Scale

We will now examine each of these in turn.

1. Transmission Technology

Transmission Technology is actually the mode (Physical Medium) and the technique which is used in a network to transmit data between the computers. Broadly speaking, there are two types of transmission technology that are in widespread use. They are as follows: Broadcast links and Point-to-point links.

- **Broadcast links**

  Broadcast networks have a single communication channel that is shared by all the machines on the network. Short messages, called packets, sent by any machine are received by all the others. An address field within the packet specifies the intended recipient.

Upon receiving a packet, a machine checks the address field. If the packet is intended for the receiving machine, that machine processes the packet; if the packet is intended for some other machine, it is just ignored.

Example, Consider someone standing at the end of a corridor with many rooms off it and shouting "Mr. Kumar, Come here. I want you". Although the packet may actually be received (heard) by many people, only Kumar responds to it. The others just ignore it.

Broadcast systems generally also allow the possibility of addressing a packet to all destinations by using a special code in the address field. When a packet with this code is transmitted, it is received and processed by every machine on the network. This mode of operation is called broadcasting.

Some broadcast systems also support transmission to a subset of the machines, something known as multicasting. One possible scheme is to reserve one bit to indicate multicasting. The remaining n - 1 address bits can hold a group number. Each machine can "subscribe" to any or all of the groups. When a packet is sent to a certain group, it is delivered to all machines subscribing to that group.

- **Point-to-point links**

Point-to-point networks consist of many connections between individual pairs of machines. To go from the source to the destination, a packet on this type of network may have to first visit one or more intermediate machines. Often multiple routes, of different lengths, are possible, so finding good ones is important in point-to-point networks.

As a general rule (although there are many exceptions), smaller, geographically localized networks tend to use broadcasting, whereas larger networks usually are point-to-point. Point-to-point transmission with one sender and one receiver is sometimes called unicasting.

2. Scale

An alternative criterion for classifying networks is their scale. In Fig 1-1 we classify multiple processor systems by their physical size. At the top are the personal area networks, networks that are meant for one person. For example, a wireless network connecting a computer with its mouse, keyboard, and printer is a personal area network. Also, a PDA that controls the user's hearing aid or pacemaker fits in this category.

Beyond the personal area networks, come longer-range networks. These can be divided into local, metropolitan, and wide area networks. Finally, the connection of two or more networks is called an internetwork. The worldwide Internet is a well-known example of an internetwork. Distance is important as a classification metric because different techniques are used at different scales. In this book we will be concerned with networks at all these scales. Below we give a brief introduction to network hardware.

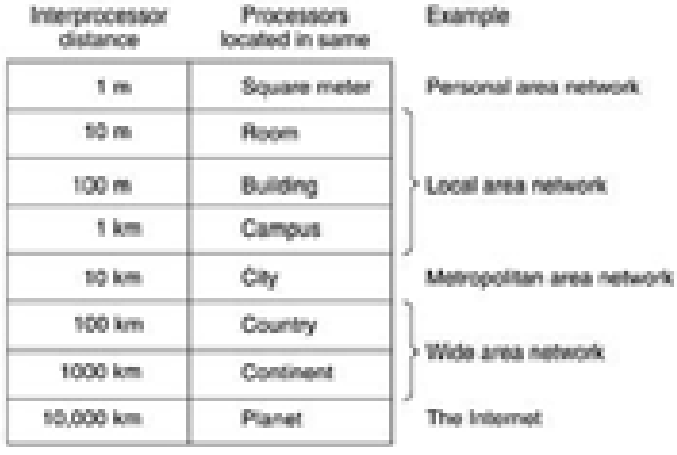


Fig 1-1 Classification of interconnected processors by scale

1.2 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: Size, Transmission technology, and 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.

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. Various topologies are possible for broadcast LANs. Fig 1-2 shows two of them.

1. Bus
2. Ring

1.2.1 Bus

In a bus (i.e., a linear cable) network, at any instant at most one machine is the master and it is allowed to transmit. All other machines are required to refrain from sending. An arbitration mechanism is needed to resolve conflicts when two or more machines want to transmit simultaneously. The arbitration mechanism may be centralized or distributed.

IEEE 802.3, popularly called Ethernet, for example, is a bus-based broadcast network with decentralized control, usually operating at 10 Mbps to 10 Gbps. Computers on an Ethernet can transmit whenever they want to; if two or more packets collide, each computer just waits a random time and tries again later.

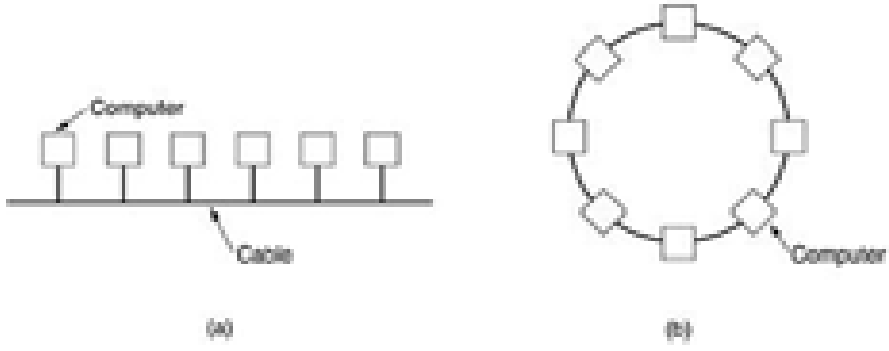


Figure 1-2. Two broadcast networks. (a) Bus. (b) Ring.

1.2.2 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 goes around 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.

1.3 Metropolitan Area Networks

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.

Starting when the Internet attracted a mass audience, the cable TV network operators began to realize that with some changes to the system, they could provide two-way Internet service in unused parts of the spectrum. At that point, the cable TV system began to morph from a way to distribute television to a metropolitan area network.

To a first approximation, a MAN might look something like the system shown in Fig. 1-3. In this figure we see both television signals and Internet being fed into the centralized head end for subsequent distribution to people's homes.

Cable television is not the only MAN. Recent developments in high-speed wireless Internet access resulted in another MAN, which has been standardized as IEEE 802.16.

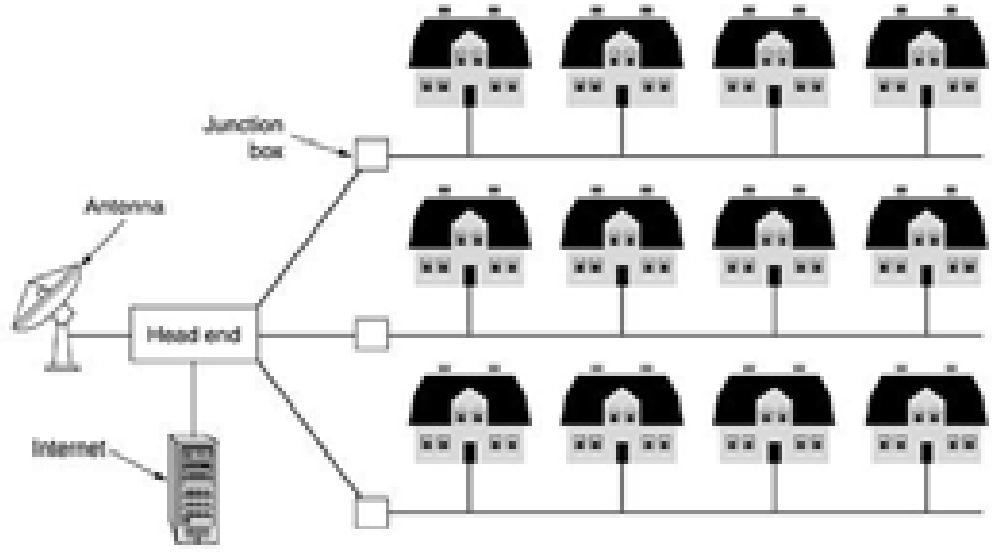


Figure 1-3. A metropolitan area network based on cable TV.

1.4 Wide Area Networks

A wide area network, or WAN, spans a large geographical area, often a country or continent. It contains a collection 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 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.

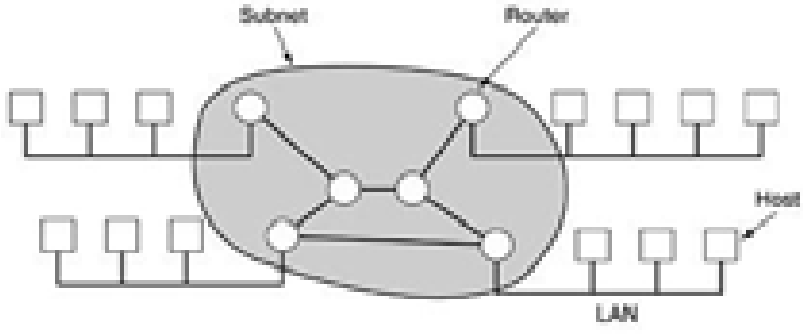


Figure 1-4. Relation between hosts on LANs and the subnet.

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, shown in Fig. 1-5.

Generally, in a packet-switched WAN, when a process on some host has a message to be sent to a process on some other host, the sending host first cuts the message into packets, each one bearing its number in the sequence.

These packets are then injected into the network one at a time in quick succession. The packets are transported individually over the network and deposited at the receiving host, where they are reassembled into the original message and delivered to the receiving process.

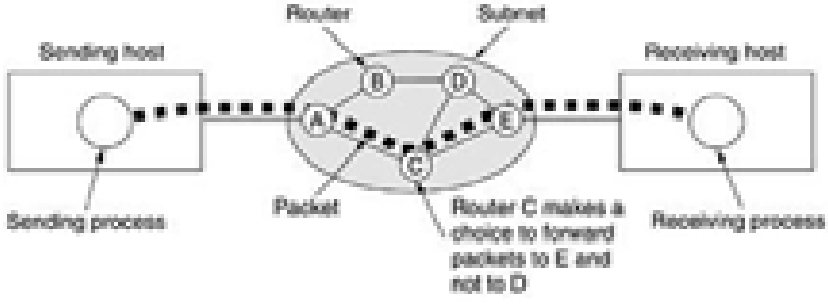


Figure 1-5. A stream of packets from sender to receiver.

In the Fig. 1-5, all the packets follow the route ACE, rather than ABDE or ACDE. In some networks all packets from a given message must follow the same route; in others each packet is routed separately. Of course, if ACE is the best route, all packets may be sent along it, even if each packet is individually routed.

Routing decisions are made locally. When a packet arrives at router A, it is up to A to decide if this packet should be sent on the line to B or the line to C. How A makes that decision is called the routing algorithm. Many of them exist.

Not all WANs are packet switched. A second possibility for a WAN is a satellite system. Each router has an antenna through which it can send and receive. All routers can hear the output from the satellite, and in some cases they can also hear the upward transmissions of their fellow routers to the satellite as well.

Sometimes the routers are connected to a substantial point-to-point subnet, with only some of them having a satellite antenna. Satellite networks are inherently broadcast and are most useful when the broadcast property is important.

1.5 Wireless Networks

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:

1. System interconnection.
2. Wireless LANs.
3. Wireless WANs.

1.5.1 System Interconnection

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. In the simplest form, system interconnection networks use the master-slave paradigm of Fig. 1-6(a).

The system unit is normally the master, talking to the mouse, keyboard, etc., as slaves. The master tells the slaves what addresses to use, when they can broadcast, how long they can transmit, what frequencies they can use, and so on.

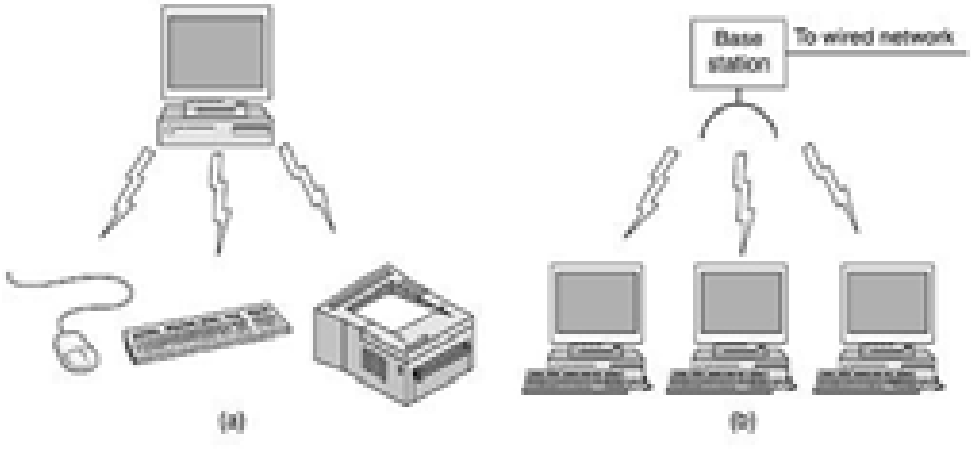


Figure 1-6. (a) Bluetooth configuration. (b) Wireless LAN.

1.5.2 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, as shown in Fig. 1-6(b).

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

1.5.3 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. A standard for it, called IEEE 802.16, has also been developed. Almost all wireless networks hook up to the wired network at some point to provide access to files, databases, and the Internet.

There are many ways these connections can be realized, depending on the circumstances. For example, in Fig. 1-7(a), we depict an airplane with a number of people using modems and seat-back telephones to call the office.

Each call is independent of the other ones. A much more efficient option, however, is the flying LAN of Fig. 1-7(b). Here each seat comes equipped with an Ethernet connector into which passengers can plug their computers.

A single router on the aircraft maintains a radio link with some router on the ground, changing routers as it flies along. This configuration is just a traditional LAN, except that its connection to the outside world happens to be a radio link instead of a hardwired line.

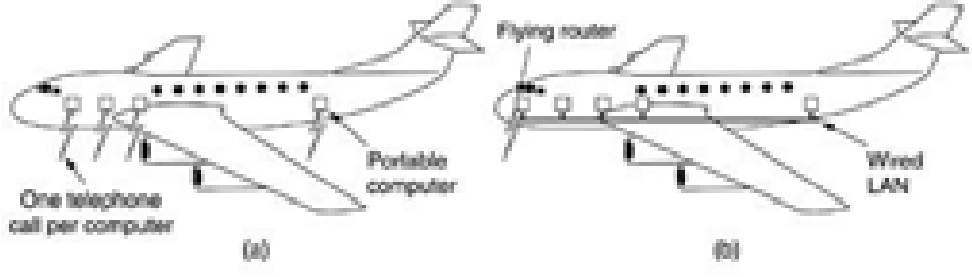


Figure 1-7. (a) Individual mobile computers. (b) A flying LAN.

1.6 Home Networks

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 (with examples) are as follows:

1. Computers (desktop PC, notebook PC, PDA, shared peripherals).

2. Entertainment (TV, DVD, VCR, camcorder, camera, stereo, MP3).
3. Telecommunications (telephone, mobile telephone, intercom, fax).
4. Appliances (microwave, refrigerator, clock, furnace, airco, lights).
5. Telemetry (utility meter, smoke/burglar alarm, thermostat, babycam).

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.

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.

1. The network and devices have to be easy to install.
2. The network and devices have to be foolproof in operation.
3. Low price is essential.
4. The main application is likely to involve multimedia, so the network needs sufficient capacity.
5. It must be possible to start out with one or two devices and expand the reach of the network gradually.
6. Security and reliability will be very important.

In short, home networking offers many opportunities and challenges. Most of them relate to the need to be easy to manage, dependable, and secure, especially in the hands of nontechnical users, while at the same time delivering high performance at low cost.

1.6.1 Internetworks

Many networks exist in the world, often with different hardware and software. People connected to one network often want to communicate with people attached to a different one. The fulfillment of this desire requires that different, and frequently incompatible networks, be connected, sometimes by means of machines called gateways to make the connection and provide the necessary translation, both in terms of hardware and software.

A collection of interconnected networks is called an internetwork or internet. These terms will be used in a generic sense, in contrast to the worldwide Internet (which is one specific internet), which we will always capitalize.

A common form of internet is a collection of LANs connected by a WAN. In fact, if we were to replace the label "subnet" in Fig. 1-6 by "WAN," nothing else in the figure would have to change. The only real technical distinction between a subnet and a WAN in this case is whether hosts are present.

If the system within the gray area contains only routers, it is a subnet; if it contains both routers and hosts, it is a WAN. The real differences relate to ownership and use. Subnets, networks, and internetworks are often confused. Subnet makes the most sense in the context of a wide area network, where it refers to the collection of routers and communication lines owned by the network operator.

As an analogy, the telephone system consists of telephone switching offices connected to one another by high-speed lines, and to houses and businesses by low-speed lines. These lines and equipment, owned and managed by the telephone company, form the subnet of the telephone system.

The telephones themselves (the hosts in this analogy) are not part of the subnet. The combination of a subnet and its hosts forms a network. In the case of a LAN, the cable and the hosts form the network. There really is no subnet. An internetwork is formed when distinct networks are interconnected. In our view, connecting a LAN and a WAN or connecting two LANs forms an internetwork, but there is little agreement in the industry over terminology in this area.

One rule of thumb is that if different organizations paid to construct different parts of the network and each maintains its part, we have an internetwork rather than a single network. Also, if the underlying technology is different in different parts (e.g., broadcast versus point-to-point), we probably have two networks.

1.7 Summary

1. There are two types of transmission technology used namely 1.Broadcast links 2.Point-to-point links.

2. The network topologies include Local Area Network, Wide Area Network and Metropolitan Area Network.

3. Wireless Networks uses Bluetooth, wireless LANs and satellite communication.

Lesson 2

Network Software

Contents

2.0 Aim

The interconnection between the homogenous and heterogeneous networks can be done by using standard set of rules which these networks obey. This lesson aims at introducing the various protocols structure and issues related with these protocols.

2.1 Introduction

The first computer networks were designed with the hardware as the main concern and the software as an afterthought. Network software is now highly structured. In the following sections we examine the software structuring technique in some detail which forms the keystone of the entire book and will occur repeatedly later on.

2.2 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. Layer n on one machine carries on a conversation with layer n on another machine. The rules and conventions used in this conversation are collectively known as the layer n protocol.

Basically, a protocol is an agreement between the communicating parties on how communication is to proceed. Violating the protocol will make

communication more difficult, if not completely impossible. A five-layer network is illustrated in Fig. 2-1.

The entities comprising the corresponding layers on different machines are called peers. The peers may be processes, hardware devices, or even human beings. In other words, it is the peers that communicate by using the protocol. 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. In Fig. 2-1, virtual communication is shown by dotted lines and physical communication by solid lines. Between each pair of adjacent layers is an interface.

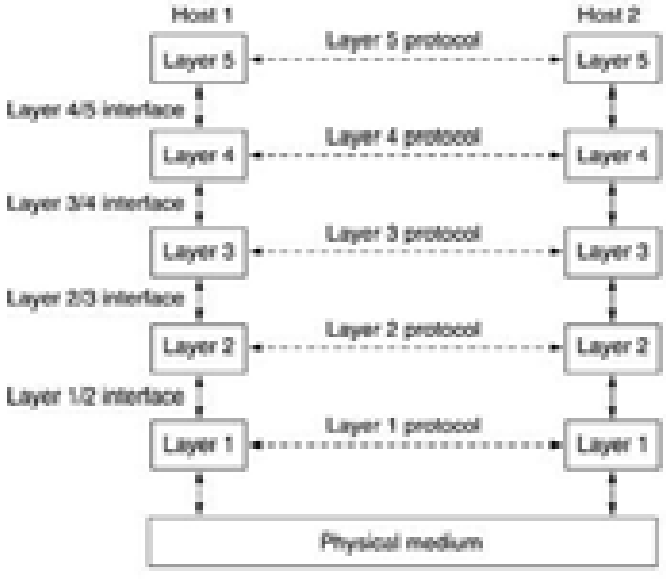


Figure 2-1. Layers, protocols, and interfaces.

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 network architecture. The specification of 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.

2.3 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.

1. **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.
2. **Set of design decisions that concern 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.
3. **Error control**
   It 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 senders which messages have been correctly received and which have not.
4. **Preserving the order of messages sent.**
   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.
5. **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.
6. **Disassembling and Reassembling**
   Another problem that must be solved at several levels is the inability of all processes to accept arbitrarily long messages. This property leads to mechanisms for disassembling, transmitting, and then reassembling messages. A related issue is the problem of what to do when processes insist on transmitting data in units that are so small that sending each one separately is inefficient. Here the solution is to gather several small

messages heading toward a common destination into a single large message and dismember the large message at the other side.

7. **Multiplexing and Demultiplexing**

   When it is inconvenient or expensive to set up a separate connection for each pair of communicating processes, the underlying layer may decide to use the same connection for multiple, unrelated conversations. As long as this multiplexing and demultiplexing is done transparently, it can be used by any layer. Multiplexing is needed in the physical layer, for example, where all the traffic for all connections has to be sent over at most a few physical circuits.

8. **Routing**

   When there are multiple paths between source and destination, a route must be chosen. Sometimes this decision must be split over two or more layers. For example, to send data from London to Rome, a high-level decision might have to be made to transit France or Germany based on their respective privacy laws. Then a low-level decision might have to made to select one of the available circuits based on the current traffic load. This topic is called routing.

2.4 Connection-Oriented and Connectionless Services

Layers can offer two different types of service to the layers above them:

1. Connection-oriented Services
2. Connectionless Services

2.4.1 Connection-Oriented Services

- ✓ 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.

2.4.2 Connection-Less Services

- ✓ 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. 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. Reliable connection-oriented service has two minor variations: message sequences and byte streams. In the former variant, the message boundaries are preserved.

When two 1024-byte messages are sent, they arrive as two distinct 1024-byte messages, never as one 2048-byte message. In the latter, the connection is simply a stream of bytes, with no message boundaries. When 2048 bytes arrive at the receiver, there is no way to tell if they were sent as one 2048-byte message, two 1024-byte messages, or 2048 1-byte messages.

If the pages of a book are sent over a network to a phototypesetter as separate messages, it might be important to preserve the message boundaries. On the other hand, when a user logs into a remote server, a byte stream from the user's computer to the server is all that is needed. Message boundaries are not relevant.

As mentioned above, for some applications, the transit delays introduced by acknowledgements are unacceptable. One such application is digitized voice traffic. It is preferable for telephone users to hear a bit of noise on the line from time to time than to experience a delay waiting for acknowledgements.

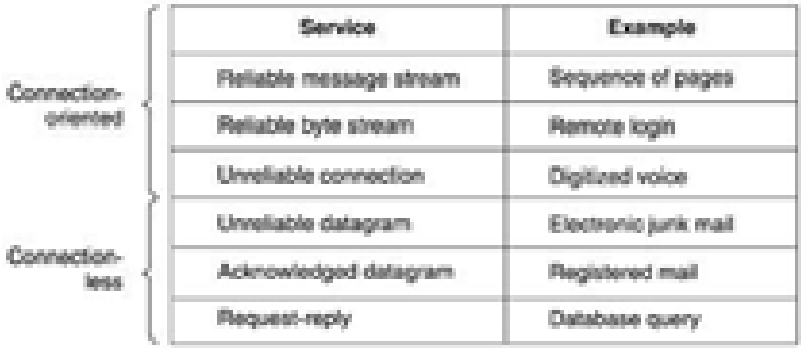
Similarly, when transmitting a video conference, having a few pixels wrong is no problem, but having the image jerk along as the flow stops to correct errors is irritating. Not all applications require connections. For example, as electronic mail becomes more common, electronic junk is becoming more common too. The electronic junk-mail sender probably does not want to go to the trouble of setting up and later tearing down a connection just to send one item. Nor is 100 percent reliable delivery essential, especially if it costs more. All that is needed is a way to send a single message that has a high probability of arrival, but no guarantee.

Unreliable (meaning not acknowledged) connectionless service is often called datagram service, in analogy with telegram service, which also does not return an acknowledgement to the sender. In other situations, the convenience of not having to establish a connection to send one short message is desired, but reliability is essential.

The acknowledged datagram service can be provided for these applications. It is like sending a registered letter and requesting a return receipt. When the receipt comes back, the sender is absolutely sure that the letter was delivered to the intended party and not lost along the way.

Still another service is the request-reply service. In this service the sender transmits a single datagram containing a request; the reply contains the

answer. For example, a query to the local library asking where Uighur is spoken falls into this category. Request-reply is commonly used to implement communication in the client-server model: the client issues a request and the server responds to it. Fig. 2-2 summarizes the types of services discussed above.



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

Figure 2-2. Six different types of service.

2.5 Service Primitives

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, consider the primitives listed in Fig. 2-3.

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

Figure 2-3. Five service primitives for implementing a simple connection-oriented service.

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.

Next, the client process executes CONNECT to establish a connection with the server. The CONNECT call needs to specify who to connect to, so it might have a parameter giving the server's address. The operating system then typically sends a packet to the peer asking it to connect, as shown by (1) in Fig. 2-4. The client process is suspended until there is a response.

When the packet arrives at the server, it is processed by the operating system there. When the system sees that the packet is requesting a connection, it checks to see if there is a listener. If so, it does two things: unblocks the listener and sends back an acknowledgement (2). The arrival of this acknowledgement then releases the client.

At this point the client and server are both running and they have a connection established. It is important to note that the acknowledgement (2) is generated by the protocol code itself, not in response to a user-level primitive. If a connection request arrives and there is no listener, the result is undefined. In some systems the packet may be queued for a short time in anticipation of a LISTEN.

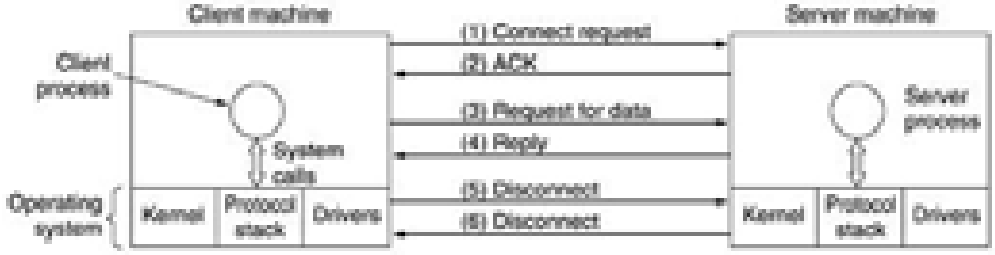


Figure 2-4. Packets sent in a simple client-server interaction on a connection-oriented network.

The obvious analogy between this protocol and real life is a customer (client) calling a company's customer service manager. The service manager starts out by being near the telephone in case it rings. Then the client places the call. When the manager picks up the phone, the connection is established.

The next step is for the server to execute RECEIVE to prepare to accept the first request. Normally, the server does this immediately upon being released from the LISTEN, before the acknowledgement can get back to the client. The RECEIVE call blocks the server. Then the client executes SEND to transmit its request (3) followed by the execution of RECEIVE to get the reply.

The arrival of the request packet at the server machine unblocks the server process so it can process the request. After it has done the work, it uses SEND to return the answer to the client (4). The arrival of this packet unblocks the client, which can now inspect the answer. If the client has additional requests, it can make them now. If it is done, it can use DISCONNECT to terminate the connection. Usually, an initial DISCONNECT is a blocking call, suspending the client and sending a packet to the server saying that the connection is no longer needed (5).

When the server gets the packet, it also issues a DISCONNECT of its own, acknowledging the client and releasing the connection. When the server's packet (6) gets back to the client machine, the client process is released and the connection is broken. In a nutshell, this is how connection-oriented communication works.

The timing can be wrong (e.g., the CONNECT is done before the LISTEN), packets can get lost, and much more. We will look at these issues in great detail later, but for the moment, Fig. 2-4 briefly summarizes how client-server communication might work over a connection-oriented network.

Given that six packets are required to complete this protocol, one might wonder why a connectionless protocol is not used instead. The answer is that in a perfect world it could be, in which case only two packets would be needed: one for the request and one for the reply.

However, in the face of large messages in direction (e.g., a megabyte file), transmission errors, and lost packets, the situation changes. If the reply consisted of hundreds of packets, some of which could be lost during transmission, how would the client know if some pieces were missing? How would the client know whether the last packet actually received was really the last packet sent? Suppose that the client wanted a second file.

How could it tell packet 1 from the second file from a lost packet 1 from the first file that suddenly found its way to the client? In short, in the real world, a simple request-reply protocol over an unreliable network is often inadequate. In lesson 3, we will study a variety of protocols in detail that overcome these and other problems. For the moment, suffice it to say that having a reliable, ordered byte stream between processes is sometimes very convenient.

2.6 The Relationship of Services to Protocols

Services and protocols are two different concepts, although they are frequently confused. This distinction is so important. 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.

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 illustrated in Fig. 2-5. In contrast, protocols relate to the packets sent between peer entities on different machines. It is important not to confuse the two concepts.

Figure 2-5. The relationship between a service and a protocol.

2.7 Summary

1. Layering of protocol is used to reduce the design complexity of the network system.
2. Connection oriented system is a reliable system resembling the telephone system.
3. Connectionless systems are unreliable and follow the conventional postal system.
4. Five basic primitive operations namely listen, connect, receive, send and disconnect are used for implementing simple connection oriented service.
5. The service differ from the protocol in the manner service is primitive or operation whereas the protocols are the rules governing these operations.

Lesson 3

Reference Models

Contents

3.0 Aim

The protocol architectures are used for reducing the complexity of the network system. Two such architectures namely OSI and TCP/IP are discussed here. Comparison between the models and their critique are also discussed in this chapter.

3.1 Introduction

In the following sections we will discuss 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.

3.2 The OSI Reference Model

The OSI model (minus the physical medium) is shown in Fig. 3-1. 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.

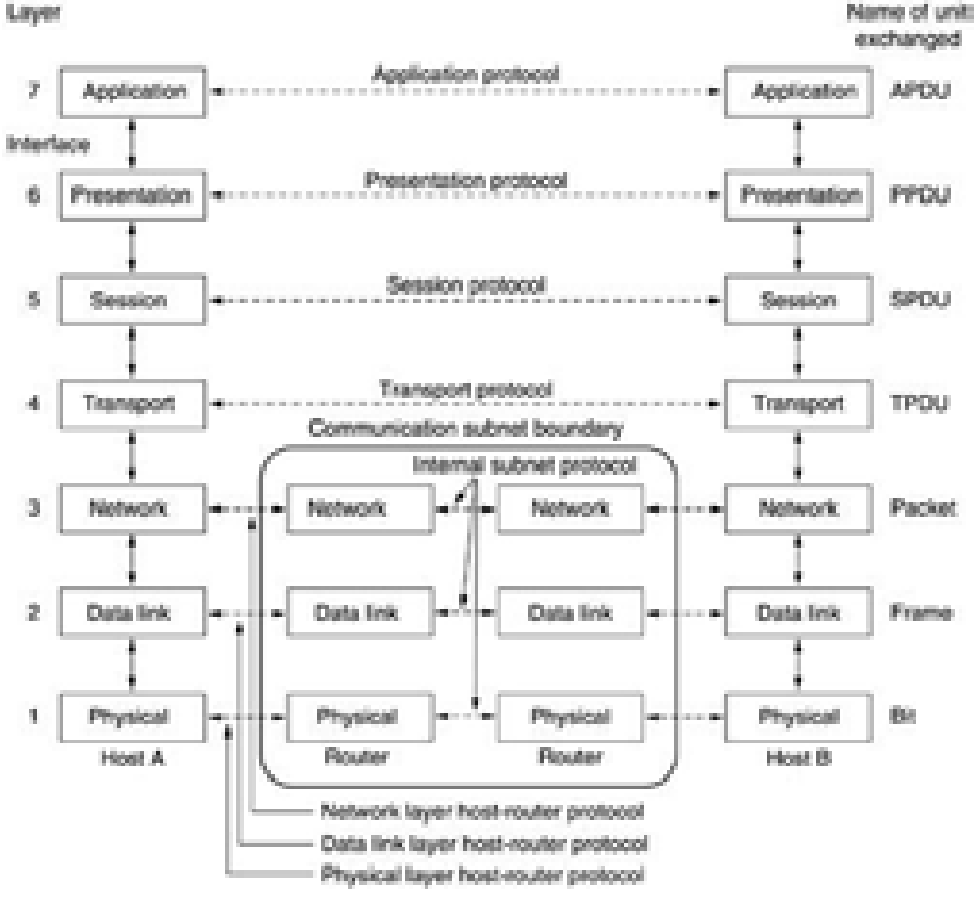


Figure 3-1. The OSI reference model.

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

1. A layer should be created where a different abstraction is needed.
2. Each layer should perform a well-defined function.
3. The function of each layer should be chosen with an eye toward defining internationally standardized protocols.
4. The layer boundaries should be chosen to minimize the information flow across the interfaces.

5. 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 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.

3.2.1 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.

3.2.2 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 transmits 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.

3.2.3 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. In broadcast networks, the routing problem is simple, so the network layer is often thin or even nonexistent.

3.2.4 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, is illustrated in Fig. 3-1.

3.2.5 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).

3.2.6 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.

3.2.7 The Application Layer

The application layer contains a variety of protocols that are commonly needed by users. One widely-used application protocol is HTTP (Hyper Text 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.

3.3 The TCP/IP Reference Model

It is 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 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 to pieces 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.

3.3.1 The Internet Layer

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 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 internet layer defines an official packet format and protocol called IP (Internet Protocol).

The job of the internet layer is to deliver IP packets where they are supposed to go. Packet routing is clearly the major issue here, as is avoiding congestion. For these reasons, it is reasonable to say that the TCP/IP internet layer is similar in functionality to the OSI network layer. Figure 3-2 shows this correspondence.

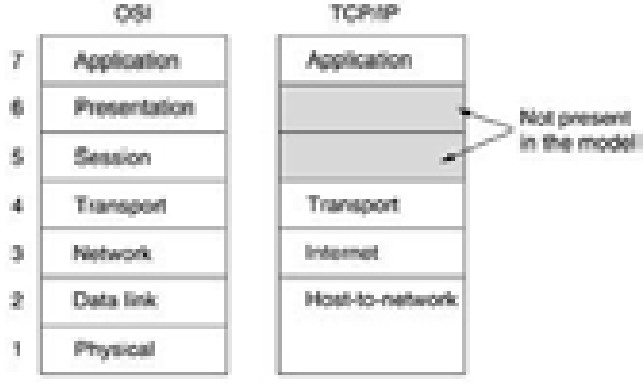


Figure 3-2. The TCP/IP reference model.

3.3.2 The Transport Layer

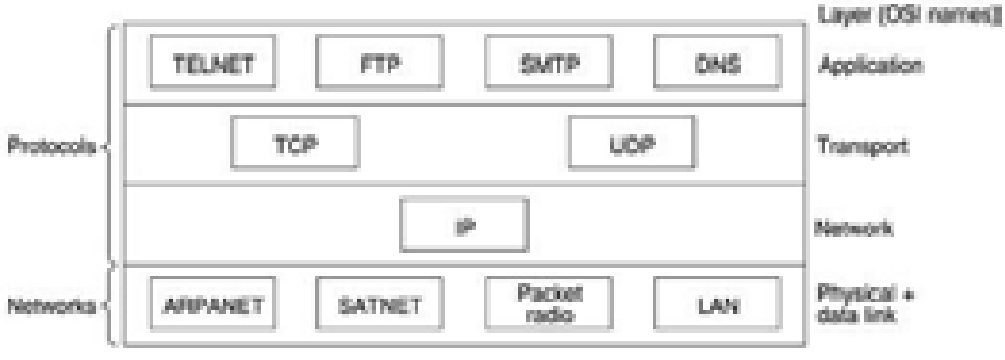
The layer above the internet layer in the TCP/IP model is now usually called the transport layer. It is designed to allow peer entities on the source and destination hosts to carry on a conversation, just as in the OSI transport layer.

Two end-to-end transport protocols have been defined here.

1. 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.

2. 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. 3-3. Since the model



was developed, IP has been implemented on many other networks.

Figure 3-3. Protocols and networks in the TCP/IP model initially.

3.3.3 The Application Layer

The TCP/IP model does not have session or presentation layers. 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), as shown in Fig. 3-3.

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.

3.3.4 The Host-to-Network Layer

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. Books and papers about the TCP/IP model rarely discuss it.

3.4 A Comparison of the OSI and TCP/IP Reference Models

Similarities

- ✓ 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.

Differences

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.

3.4.1 About OSI

Three concepts are central to the OSI model. Probably the biggest contribution of the OSI model is to make the distinction between these three concepts explicit.

1. Services.
2. Interfaces.
3. Protocols.

1. Services

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.

2. Interfaces

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.

3. Protocols

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 form the object's interface. The code internal to the object is its protocol and is not visible or of any concern outside the object.

3.4.2 About TCP/IP

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.

As a consequence, the protocols in the OSI model are better hidden than in the TCP/IP model and can be replaced relatively easily as the technology changes. Being able to make such changes is one of the main purposes of having layered protocols in the first place.

An obvious difference between the two models is the number of layers: the OSI model has seven layers and the TCP/IP has four layers. Both have (inter)network, transport, and application layers, but the other layers are different.

Another difference is in the area of connectionless versus connection-oriented communication. The OSI model supports both connectionless and connection-oriented communication in the network layer, but only connection-oriented communication in the transport layer, where it counts (because the transport service is visible to the users).

The TCP/IP model has only one mode in the network layer (connectionless) but supports both modes in the transport layer, giving the users a choice. This choice is especially important for simple request-response protocols.

3.5 OSI Model and Protocols - A Critique

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.

Some of the critics about OSI Model and Protocols can be summarized under the two topics as stated below.

1. Bad technology.
2. Bad implementations.

3.5.1 Bad Technology

- The reason that OSI is criticized 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.

- The OSI model, along with the associated service definitions and protocols, is extraordinarily complex.
- When piled up, the printed standards occupy a significant fraction of a meter of paper.
- They are also difficult to implement and inefficient in operation.
- In addition to being incomprehensible, another problem with OSI is that some functions, such as addressing, flow control, and error control, reappear again and again in each layer. Saltzer et al. (1984), for example, have pointed out that to be effective, error control must be done in the highest layer, so that repeating it over and over in each of the lower layers is often unnecessary and inefficient.

3.5.2 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.
- In contrast, one of the first implementations of TCP/IP was part of Berkeley UNIX and was quite good (not to mention, free).
- People began using it quickly, which led to a large user community, which led to improvements, which led to an even larger community.
- Here the spiral was upward instead of downward.

3.6 TCP/IP Reference Model - A Critique

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.

- 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. The virtual terminal protocol, TELNET, for example, was designed for a ten-character per second mechanical Teletype terminal. It knows nothing of graphical user interfaces and mice. Nevertheless, 25 years later, it is still in widespread use.

3.7 Summary

1. This model is based on a proposal developed by the International Standards Organization (ISO) as a first step toward international standardization of the protocols
2. The OSI model has seven layers with each layer performing well defined functions.
3. The TCP/IP model was defined by ARPANET and eventually used in the current internet setup.
4. The OSI and TCP/IP models differs each other at Services, Interfaces and Protocols level.
5. The bad technology and bad implementation of the reference models becomes the critique factor of each.

UNIT - II

Lesson 4

Guided Transmission Media

Contents

4.0 Aim

This chapter deals with the various transmission media used in the computer networks ranging from magnetic media to fiber optics. The chapter also deals with the efficiency and implementation issues of all these media.

4.1 Introduction

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.

4.2 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.

If we now look at cost, we get a similar picture. The cost of an Ultrium tape is around $40 when bought in bulk. A tape can be reused at least ten times, so the tape cost is maybe $4000 per box per usage. Add to this another $1000 for shipping (probably much less), and we have a cost of roughly $5000 to ship 200 TB. This amounts to shipping a gigabyte for under 3 cents. No network can beat that. The moral of the story is:

"Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway."

4.3 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. 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. Starting around 1988, the more advanced category 5 twisted pairs were introduced.

They are similar to category 3 pairs, but with more twists per centimeter, which results in less crosstalk and a better-quality signal over longer distances, making them more suitable for high-speed computer communication. Up-and-coming categories are 6 and 7, which are capable of handling signals with bandwidths of 250 MHz and 600 MHz, respectively (versus a mere 16 MHz and 100 MHz for categories 3 and 5, respectively).

All of these wiring types are often referred to as UTP (Unshielded Twisted Pair), to contrast them with the bulky, expensive, shielded twisted pair cables IBM introduced in the early 1980s, but which have not proven popular outside of IBM installations. Twisted pair cabling is illustrated in Fig. 4-1

Figure 4-1. (a) Category 3 UTP. (b) Category 5 UTP.

4.4 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 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. A cutaway view of a coaxial cable is shown in Fig. 4-2.

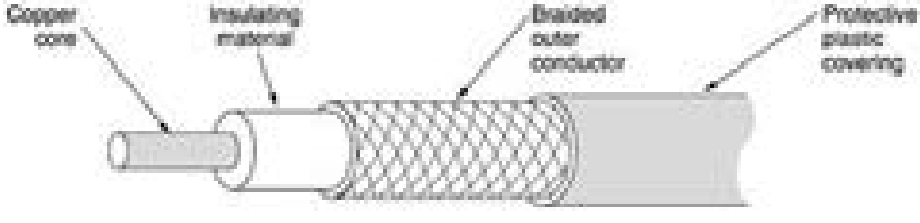


Figure 4-2. A coaxial cable.

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.

4.5 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.

An optical transmission system has three key components:

1. The Light Source
2. The Transmission Medium
3. The Detector

Conventionally, a pulse of light indicates a 1 bit and the absence of light indicates a 0 bit. The transmission medium is an ultra-thin fiber of glass. The detector generates an electrical pulse when light falls on it. By attaching a light source to one end of an optical fiber and a detector to the other, we have a unidirectional data transmission system that accepts an electrical signal,

converts and transmits it by light pulses, and then reconverts the output to an electrical signal at the receiving end.

This transmission system would leak light and be useless in practice except for an interesting principle of physics. When a light ray passes from one medium to another, for example, from fused silica to air, the ray is refracted (bent) at the silica/air boundary, as shown in Fig. 4-3(a). Here we see a light ray incident on the boundary at an angle θ_1 emerging at an angle θ_1.

The amount of refraction depends on the properties of the two media (in particular, their indices of refraction). For angles of incidence above a certain critical value, the light is refracted back into the silica; none of it escapes into the air. Thus, a light ray incident at or above the critical angle is trapped inside the fiber, as shown in Fig. 4-3(b), and can propagate for many kilometers with virtually no loss.

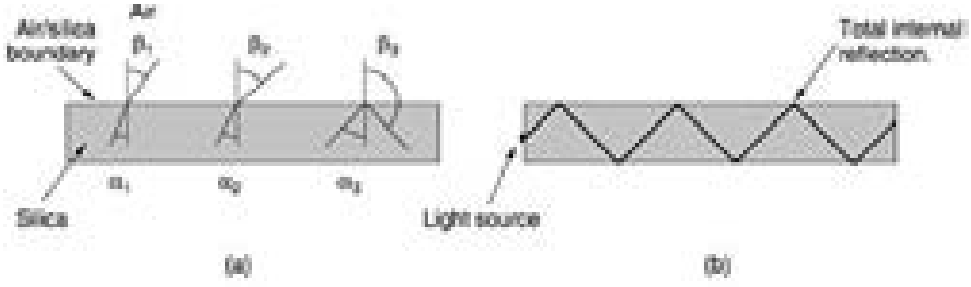


Figure 4-3. (a) Three examples of a light ray from inside a silica fiber impinging on the air/silica boundary at different angles. (b) Light trapped by total internal reflection.

The sketch of Fig. 4-3(b) shows only one trapped ray, but since any light ray incident on the boundary above the critical angle will be reflected internally, many different rays will be bouncing around at different angles.

Each ray is said to have a different mode, so a fiber having this property is called a multimode fiber. However, if the fiber's diameter is reduced to a few wavelengths of light, the fiber acts like a wave guide, and the light can propagate only in a straight line, without bouncing, yielding a single-mode fiber.

Single-mode fibers are more expensive but are widely used for longer distances. Currently available single-mode fibers can transmit data at 50 Gbps for 100 km without amplification. Even higher data rates have been achieved in the laboratory for shorter distances.

4.5.1 Transmission of Light through Fiber

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 not more than 1 mm thick 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, the attenuation is shown in Fig. 4-4 in decibels per linear kilometer of fiber. The attenuation in decibels is given by the formula

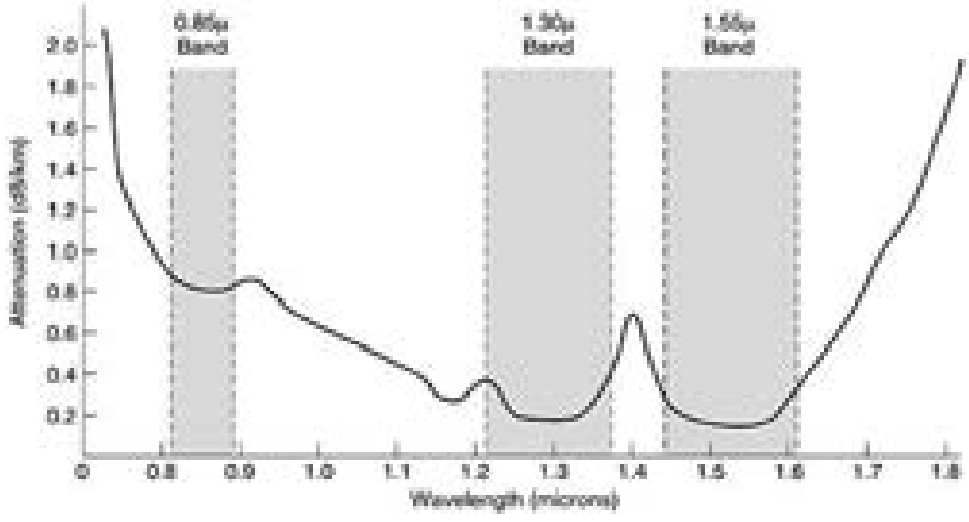


Figure 4-4. Attenuation of light through fiber in the infrared region.

For example, a factor of two loss gives an attenuation of 10 $\log_{10}$ 2 = 3 dB. The figure shows the near infrared part of the spectrum, which is what is used in practice. Visible light has slightly shorter wavelengths, from 0.4 to 0.7 microns (1 micron is 10^{-6} meters). The true metric purist would refer to these wavelengths as 400 nm to 700 nm, but we will stick with traditional usage.

Three wavelength bands are used for optical communication. They are centered at 0.85, 1.30, and 1.55 microns, respectively. The last two have good attenuation properties (less than 5 percent loss per kilometer). The 0.85 micron band has higher attenuation, but at that wavelength the lasers and electronics can be made from the same material (gallium arsenide).

All three bands are 25,000 to 30,000 GHz wide. Light pulses sent down a fiber spread out in length as they propagate. This spreading is called chromatic dispersion. The amount of it is wavelength dependent. One way to keep these spread-out pulses from overlapping is to increase the distance between them, but this can be done only by reducing the signaling rate. Fortunately, it has been discovered that by making the pulses in a special shape related to the reciprocal of the hyperbolic cosine, nearly all the dispersion effects cancel out, and it is possible to send pulses for thousands of kilometers without appreciable shape distortion. These pulses are called solitons. A considerable amount of research is going on to take solitons out of the lab and into the field.

4.5.2 Fiber Cables

Fiber optic cables are similar to coax, except without the braid. Figure 4-5(a) 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.

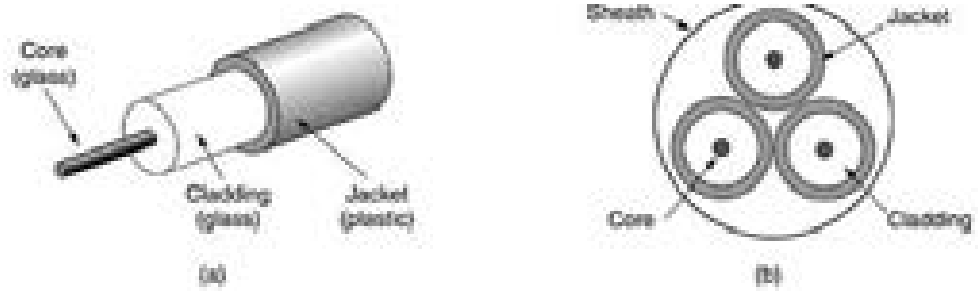


Figure 4-5. (a) Side view of a single fiber. (b) End view of a sheath with three fibers.

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. Figure 4-5(b) shows a sheath with three fibers. 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 sea plow. 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.

For all three kinds of splices, reflections can occur at the point of the splice, and the reflected energy can interfere with the signal.

Two kinds of light sources are typically used to do the signaling

1. LEDs (Light Emitting Diodes)
2. Semi-conductor Lasers

They have different properties, as shown in Fig. 4-6. They can be tuned in wavelength by inserting Fabry-Perot or Mach-Zehnder interferometers between the source and the fiber. Fabry-Perot interferometers are simple resonant cavities consisting of two parallel mirrors. The light is incident perpendicular to the mirrors.

The length of the cavity selects out those wavelengths that fit inside an integral number of times. Mach-Zehnder interferometers separate the light into two beams. The two beams travel slightly different distances. They are recombined at the end and are in phase for only certain wavelengths.

Item	LED	Semiconductor laser
Data rate	Low	High
Fiber type	Multimode	Multimode or single mode
Distance	Short	Long
Lifetime	Long life	Short life
Temperature sensitivity	Minor	Substantial
Cost	Low cost	Expensive

Figure 4-6. A comparison of semiconductor diodes and LEDs as light sources.

The receiving end of an optical fiber consists of a photodiode, which gives off an electrical pulse when struck by light. The typical response time of a photodiode is 1 nsec, which limits data rates to about 1 Gbps.

Thermal noise is also an issue, so a pulse of light must carry enough energy to be detected. By making the pulses powerful enough, the error rate can be made arbitrarily small.

4.5.3 Fiber Optic Networks

Fiber optics can be used for LANs as well as for long-haul transmission, although tapping into it is more complex than connecting to an Ethernet. One way around the problem is to realize that a ring network is really just a collection of point-to-point links, as shown in Fig. 4-7.

The interface at each computer passes the light pulse stream through to the next link and also serves as a T junction to allow the computer to send and accept messages.

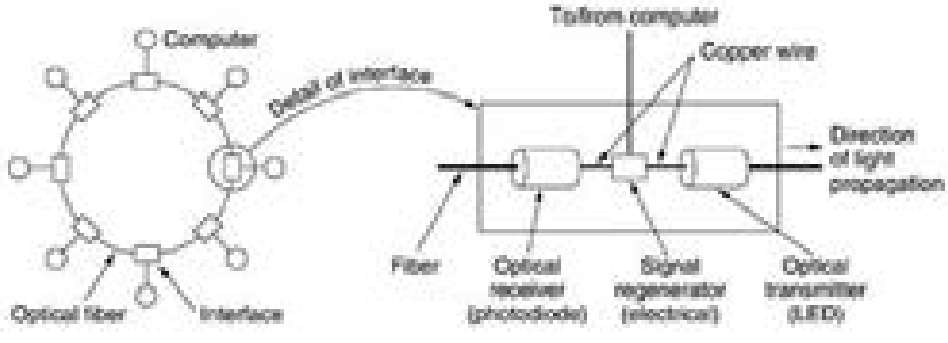


Figure 4-7. A fiber optic ring with active repeaters.

Two types of interfaces are used. A passive interface consists of two taps fused onto the main fiber. One tap has an LED or laser diode at the end of it (for transmitting), and the other has a photodiode (for receiving).

The tap itself is completely passive and is thus extremely reliable because a broken LED or photodiode does not break the ring. It just takes one

computer off-line. The other interface type, shown in Fig. 4-7, is the active repeater.

The incoming light is converted to an electrical signal, regenerated to full strength if it has been weakened, and retransmitted as light. The interface with the computer is an ordinary copper wire that comes into the signal regenerator. Purely optical repeaters are now being used, too. These devices do not require the optical to electrical to optical conversions, which mean they can operate at extremely high bandwidths.

If an active repeater fails, the ring is broken and the network goes down. On the other hand, since the signal is regenerated at each interface, the individual computer-to-computer links can be kilometers long, with virtually no limit on the total size of the ring. The passive interfaces lose light at each junction, so the number of computers and total ring length are greatly restricted.

A ring topology is not the only way to build a LAN using fiber optics. It is also possible to have hardware broadcasting by using the passive star construction of Fig. 4-8. In this design, each interface has a fiber running from its transmitter to a silica cylinder, with the incoming fibers fused to one end of the cylinder.

Similarly, fibers fused to the other end of the cylinder are run to each of the receivers. Whenever an interface emits a light pulse, it is diffused inside the passive star to illuminate all the receivers, thus achieving broadcast. In effect, the passive star combines all the incoming signals and transmits the merged result on all lines.

Since the incoming energy is divided among all the outgoing lines, the number of nodes in the network is limited by the sensitivity of the photodiodes.

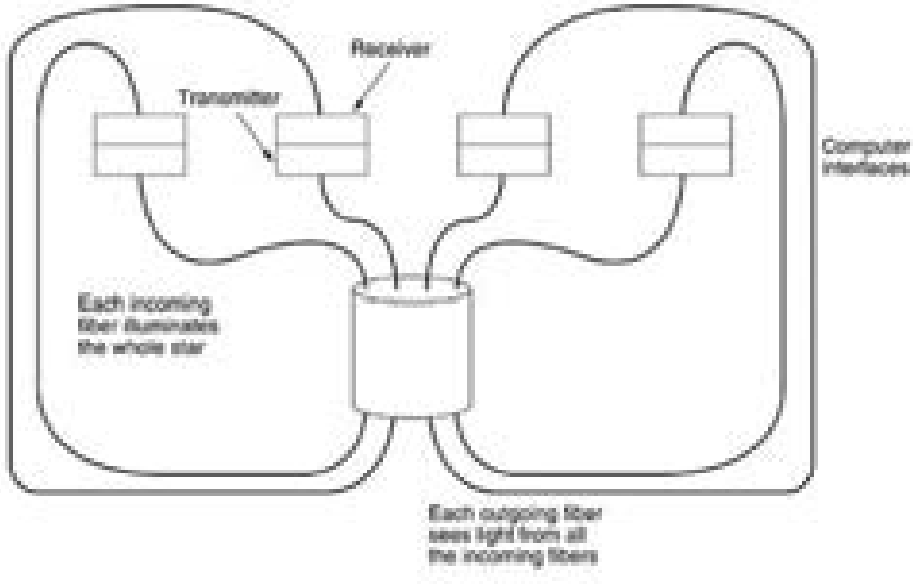


Figure 4-8. A passive star connection in a fiber optics network.

4.5.4 Comparison of Fiber Optics and Copper Wire

It is instructive to compare fiber to copper. Fiber has many advantages.

✓ It can handle much higher bandwidths than copper. This alone would require its use in high-end networks.

✓ Due to the low attenuation, repeaters are needed only about every 50 km on long lines, versus about every 5 km for copper, a substantial cost saving.

✓ Fiber also has the advantage of not being affected by power surges, electromagnetic interference, or power failures. Nor is it affected by corrosive chemicals in the air, making it ideal for harsh factory environments.

✓ Telephone companies like fiber for a different reason: it is thin and lightweight.

✓ Many existing cable ducts are completely full, so there is no room to add new capacity.

✓ Removing all the copper and replacing it by fiber empties the ducts, and the copper has excellent resale value to copper refiners who see it as very high grade ore.

✓ Also, fiber is much lighter than copper.

✓ One thousand twisted pairs 1 km long weigh 8000 kg. Two fibers have more capacity and weigh only 100 kg, which greatly reduces the need for expensive mechanical support systems that must be maintained. For new routes, fiber wins hands down due to its much lower installation cost.

✓ Finally, fibers do not leak light and are quite difficult to tap.

These properties give fiber excellent security against potential wire tappers.

Fiber has many disadvantages too.

✓ On the downside, fiber is a less familiar technology requiring skills not all engineers have, and fibers can be damaged easily by being bent too much.

✓ Since optical transmission is inherently unidirectional, two-way communication requires either two fibers or two frequency bands on one fiber.

✓ Finally, fiber interfaces cost more than electrical interfaces.

4.6 Summary

1. Magnetic media is the most common ways to transport data from one computer to another by writing them onto magnetic tape or removable media and physically transport the tape or disks to the destination machine, and read them back in again.
2. Twisted pair contains two insulted wires which follows the conventional telephone system.
3. Coaxial cable has better shielding than twisted pairs, so it can span longer distances at higher speeds.
4. Optical communication serves as the fastest communication media using light as the transmission medium.

Lesson 5

Wireless Transmission

Contents

5.0 Aim

The advent of the wireless communication by means of radio transmission, electro magnetic spectrum and light sources has great impact over the transmission technology and cost of the media for transmission. This chapter deals with the various wireless transmission techniques with its performance.

5.1 Introduction

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.

5.2 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 λ (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 3×10^8 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, λ, and c (in vacuum) is

$$\lambda f = c \quad \text{.....Eq 5.1}$$

Since c is a constant, if we know f, we can find λ, and vice versa. As a rule of thumb, when λ is in meters and f is in MHz, $\lambda f \approx 300$. 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 electromagnetic spectrum is shown in Fig. 5-1.

The radio, microwave, infrared, and visible light portions of the spectrum can all be used for transmitting information by modulating the amplitude, frequency, or phase of the waves. Ultraviolet light, X-rays, and gamma rays would be even better, due to their higher frequencies, but they are hard to produce and modulate, do not propagate well through buildings, and are dangerous to living things.

The bands listed at the bottom of Fig. 5-1 are the official ITU names and are based on the wavelengths, so the LF band goes from 1 km to 10 km (approximately 30 kHz to 300 kHz). The terms LF, MF, and HF refer to low, medium, and high frequency, respectively. Clearly, when the names were assigned, nobody expected to go above 10 MHz, so the higher bands were later named the Very, Ultra, Super, Extremely, and Tremendously High Frequency bands.

Beyond that there are no names, but Incredibly, Astonishingly, and Prodigiously high frequency (IHF, AHF, and PHF) would sound nice.

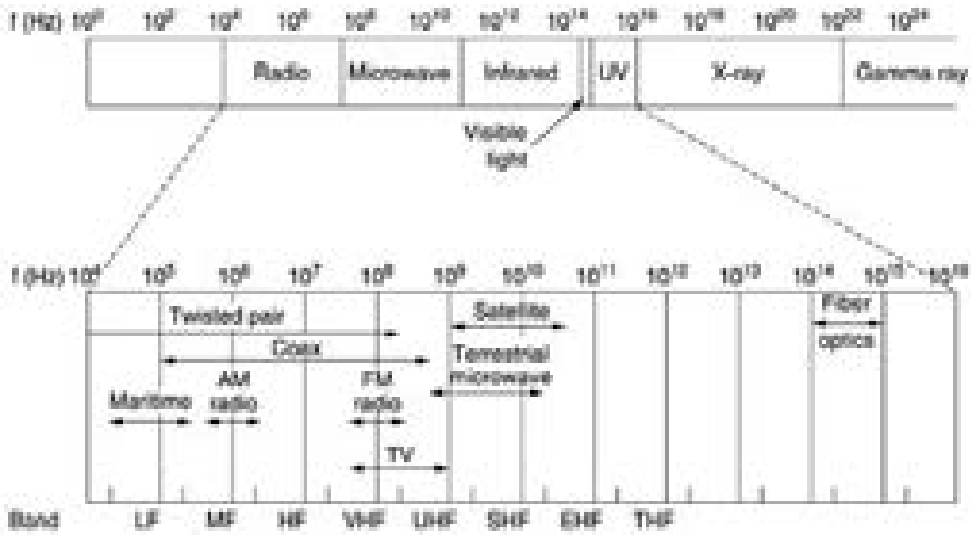


Figure 5-1. The electromagnetic spectrum and its uses for communication.

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.

From Fig. 5-1, it should now be obvious why networking people like fiber optics so much. If we solve Eq. (5-2) for f and differentiate with respect to □, we get

$$\frac{df}{d\lambda} = \frac{c}{\lambda^2} \text{ .....Eq.5.2}$$

If we now go to finite differences instead of differentials and only look at absolute values, we get

$$\Delta f = \frac{c}{\lambda^2} \text{ ..... Eq 5.3}$$

Thus, given the width of a wavelength band, $\Delta\lambda$, we can compute the corresponding frequency band, Δf, and from that the data rate the band can produce.

The wider the band, the higher is the data rate. As an example, consider the 1.30-micron band of Fig.4-4 Here we have $\lambda = 1.3 \times 10^{-6}$ and $\Delta\lambda = 0.17 \times 10^{-6}$, so Δf is about 30 THz. At, say, 8 bits/Hz, we get 240 Tbps. Most transmissions use a narrow frequency band (i.e., $\Delta f/f << 1$) 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.

The other form of spread spectrum, direct sequence spread spectrum, which spreads the signal over a wide frequency band, is also gaining popularity in the commercial world. In particular, some second-generation mobile phones use it, and it will become dominant with the third generation, thanks to its good spectral efficiency, noise immunity, and other properties. Some wireless LANs also use it.

5.3 Radio Transmission

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 omnidirectional, meaning that they travel in all directions from the source, so the transmitter and receiver do not have to be carefully aligned physically. Sometimes omnidirectional radio is good, but sometimes it is bad. 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, as illustrated in Fig 5-2(a).

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

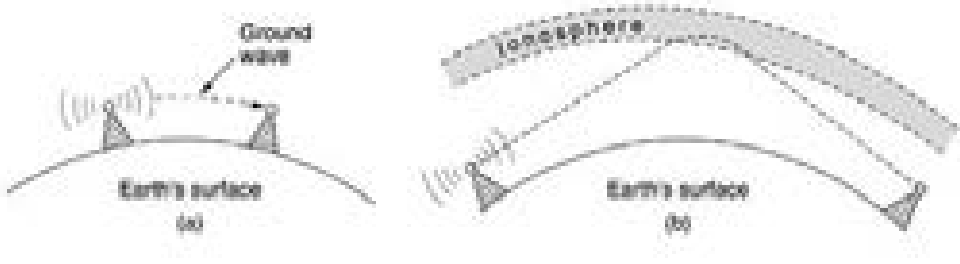


Figure 5-2. (a) In the VLF, LF, and MF bands, radio waves follow the curvature of the earth. (b) In the HF band, they bounce off the ionosphere.

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, as shown in Fig 5-2(b)

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.

5.4 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.

Consequently, repeaters are needed periodically. The higher the towers are, the farther apart they can be. he 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. Unlike radio waves at lower frequencies, microwaves do not pass through buildings well.

In addition, even though the beam may be well focused at the transmitter, there is still some divergence in space. Some waves may be refracted off low-lying atmospheric layers and may take slightly longer to arrive than the direct waves. The delayed waves may arrive out of phase with the direct wave and thus cancel the signal. This effect is called multipath fading and is often a serious problem.

It is weather and frequency dependent. Some operators keep 10 percent of their channels idle as spares to switch on when multipath fading wipes out some frequency band temporarily. The demand for more and more spectrum drives operators to yet higher frequencies. Bands up to 10 GHz are now in routine use, but at about 4 GHz a new problem sets in: absorption by water.

These waves are only a few centimeters long and are absorbed by rain. This effect would be fine if one were planning to build a huge outdoor microwave oven for roasting passing birds, but for communication, it is a severe problem. As with multipath fading, the only solution is to shut off links that are being rained on and route around them.

In summary, microwave communication is so widely used for long-distance telephone communication, mobile phones, television distribution, and other uses that a severe shortage of spectrum has developed.

It has several significant advantages over fiber.

- ✓ The main one is that no right of way is needed, and by buying a small plot of ground every 50 km and putting a microwave tower on it, one can bypass the telephone system and communicate directly.
- ✓ Microwave is also relatively inexpensive. Putting up two simple towers (may be just big poles with four guy wires) and putting antennas on each one may be cheaper than burying 50 km of fiber through a congested urban area or up over a mountain, and it may also be cheaper than leasing the telephone company's fiber, especially if the telephone company has not yet even fully paid for the copper it ripped out when it put in the fiber.

Electromagnetic Spectrum

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 spectrums. 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).

Even when a piece of spectrum has been allocated to some use, such as mobile phones, there is the additional issue of which carrier is allowed to use which frequencies.

Three algorithms were widely used in the past.

1. **The Beauty Contest Algorithm**

   It requires each carrier to explain why its proposal serves the public interest best. Government officials then decide which of the nice stories they enjoy most. Having some government official award property worth billions of dollars to his favorite company often leads to bribery, corruption, nepotism, and worse.

2. **Algorithm 2 - Holding a lottery among the interested companies.**

   The problem with that idea is that companies with no interest in using the spectrum can enter the lottery. If, say, a fast food restaurant or shoe store chain wins, it can resell the spectrum to a carrier at a huge profit and with no risk.

3. **Algorithm 3 - Auctioning off the bandwidth to the highest bidder.**

   When England auctioned off the frequencies needed for third-generation mobile systems in 2000, they expected to get about $4 billion.

They actually received about $40. This event switched on nearby governments' greedy bits and inspired them to hold their own auctions. It worked, but it also left some of the carriers with so much debt that they are close to bankruptcy. Even in the best cases, it will take many years to recoup the licensing fee.

A Different Approach

A completely different approach to allocating frequencies is to not allocate them at all. Just let everyone transmit at will but regulate the power used so that stations have such a short range they do not interfere with each other.

Accordingly, most governments have set aside some frequency bands, called the ISM (Industrial, Scientific, Medical) bands for unlicensed usage. Garage door openers, cordless phones, radio-controlled toys, wireless mice, and numerous other wireless household devices use the ISM bands. To minimize interference between these uncoordinated devices, the FCC mandates that all devices in the ISM bands use spread spectrum techniques. Similar rules apply in other countries

The location of the ISM bands varies somewhat from country to country. In the United States, for example, devices whose power is under 1 watt can use the bands shown in Fig 5-3 without requiring a FCC license. The 900-MHz band works best, but it is crowded and not available worldwide.

The 2.4-GHz band is available in most countries, but it is subject to interference from microwave ovens and radar installations. Bluetooth and some of the 802.11 wireless LANs operate in this band. The 5.7-GHz band is new and relatively undeveloped, so equipment for it is expensive, but since 802.11a uses it, it will quickly become more popular.

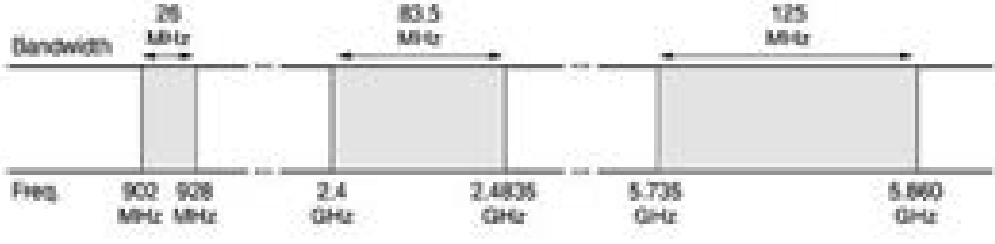


Figure 5-3. The ISM bands in the United States.

5.5 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.

5.6 Lightwave 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. Lasers have the following disadvantage too.

A conference was arranged 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.

After the conference, the organizers discovered the problem. Heat from the sun during the day time caused convection currents to rise up from the roof of the building, as shown in Fig. 5-4. This turbulent air diverted the beam and made it dance around the detector. Atmospheric "seeing" like this makes the stars twinkle (which is why astronomers put their telescopes on the tops of mountains—to get above as much of the atmosphere as possible). It is also responsible for shimmering roads on a hot day and the wavy images seen when one looks out above a hot radiator.

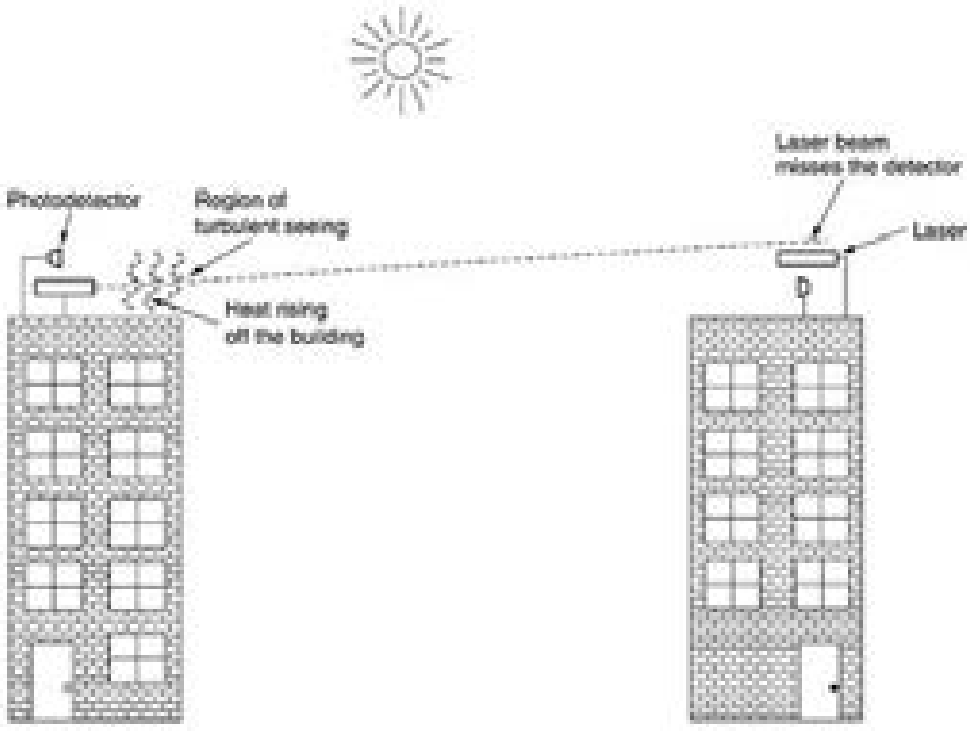


Figure 5-4. Convection currents can interfere with laser communication systems. A bidirectional system with two lasers is pictured here.

5.7 Summary

1. The basic principle of the wireless transmission is that 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

2. Radio waves are easy to generate, can travel long distances, and can penetrate buildings easily.

3. Microwave transmission uses narrowly focused wave which have a high signal-noise ratio with the antennas aligned to each other.

4. Unguided infrared and millimeter waves and are widely used for short-range communication.

5. Coherent optical signaling using lasers act as the media for lightwave transmission. It offers high bandwidth at very low cost.

Lesson 6

Communication Satellites

Contents

6.0 Aim

This lesson deals with one of the important means of wireless communication satellites. The properties of the satellite and the various techniques used in the communication satellite are discussed here. Comparison of the communication satellites with other wireless communication techniques are also dealt later in this lesson.

6.1 Introduction

In the 1950s and early 1960s, people tried to set up communication systems by bouncing signals off metalized 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.

Properties of Communication Satellites

- 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 is 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. These regions and some of their properties are illustrated in Fig. 6-1.

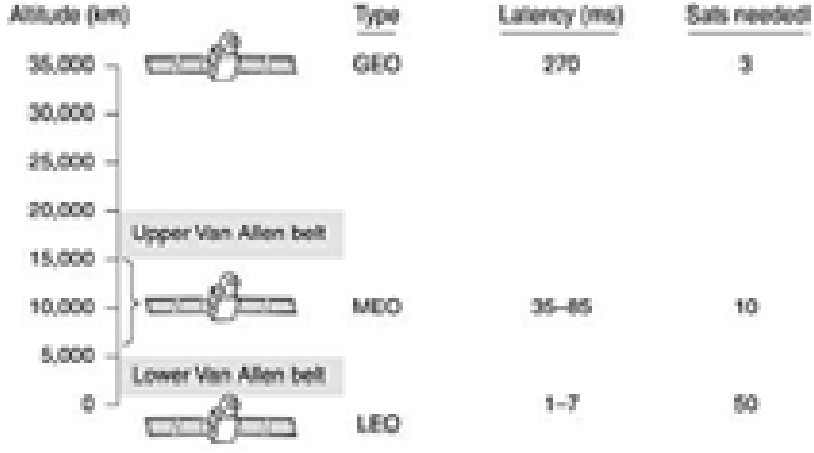


Figure 6-1. Communication satellites and some of their properties, including altitude above the earth, round-trip delay time, and number of satellites needed for global coverage.

6.2 Geostationary Satellites

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.

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. Orbit slots are not the only bone of contention. Frequencies are, too, because the downlink transmissions interfere with existing microwave users. Consequently, ITU has allocated certain frequency bands to satellite users. The main ones are listed in Fig 6-2.

The C band was the first to be designated for commercial satellite traffic. Two frequency ranges are assigned in it, the lower one for downlink traffic (from the satellite) and the upper one for uplink traffic (to the satellite). To allow traffic to go both ways at the same time, two channels are required, one going each way.

These bands are already overcrowded because they are also used by the common carriers for terrestrial microwave links. The L and S bands were added by international agreement in 2000. However, they are narrow and crowded.

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

Figure 6-2. The principal satellite bands.

The next highest band available to commercial telecommunication carriers is the Ku (K under) band. This band is not (yet) congested, and at these frequencies, satellites can be spaced as close as 1 degree. However, another problem exists: rain. Water is an excellent absorber of these short microwaves.

Fortunately, heavy storms are usually localized, so using several widely separated ground stations instead of just one circumvents the problem but at the price of extra antennas, extra cables, and extra electronics to enable rapid switching between stations. Bandwidth has also been allocated in the Ka (K above) band for commercial satellite traffic, but the equipment needed to use it is still expensive.

In addition to these commercial bands, many government and military bands also exist. A modern satellite has around 40 transponders, each with an 80-MHz bandwidth. Usually, each transponder operates as a bent pipe, but recent satellites have some on-board processing capacity, allowing more sophisticated operation. In the earliest satellites, the division of the transponders into channels was static: the bandwidth was simply split up into fixed frequency bands.

Nowadays, each transponder beam is divided into time slots, with various users taking turns. The first geostationary satellites had a single spatial beam that illuminated about 1/3 of the earth's surface, called its footprint. With the enormous decline in the price, size, and power requirements of microelectronics, a much more sophisticated broadcasting strategy has become possible.

Each satellite is equipped with multiple antennas and multiple transponders. Each downward beam can be focused on a small geographical area, so multiple upward and downward transmissions can take place simultaneously. Typically, these so-called spot beams are elliptically shaped, and can be as small as a few hundred km in diameter.

A communication satellite for the United States typically has one wide beam for the contiguous 48 states, plus spot beams for Alaska and Hawaii. A new development in the communication satellite world is the development of low-cost micro stations, sometimes called VSATs (Very Small Aperture Terminals) (Abramson, 2000). These tiny terminals have 1-meter or smaller antennas (versus 10 m for a standard GEO antenna) and can put out about 1 watt of power.

The uplink is generally good for 19.2 kbps, but the downlink is more often 512 kbps or more. Direct broadcast satellite television uses this technology for one-way transmission. In many VSAT systems, the micro

stations do not have enough power to communicate directly with one another (via the satellite, of course).

Instead, a special ground station, the hub, with a large, high-gain antenna is needed to relay traffic between VSATs, as shown in Fig. 6-3. In this mode of operation, either the sender or the receiver has a large antenna and a powerful amplifier. The trade-off is a longer delay in return for having cheaper end-user stations.

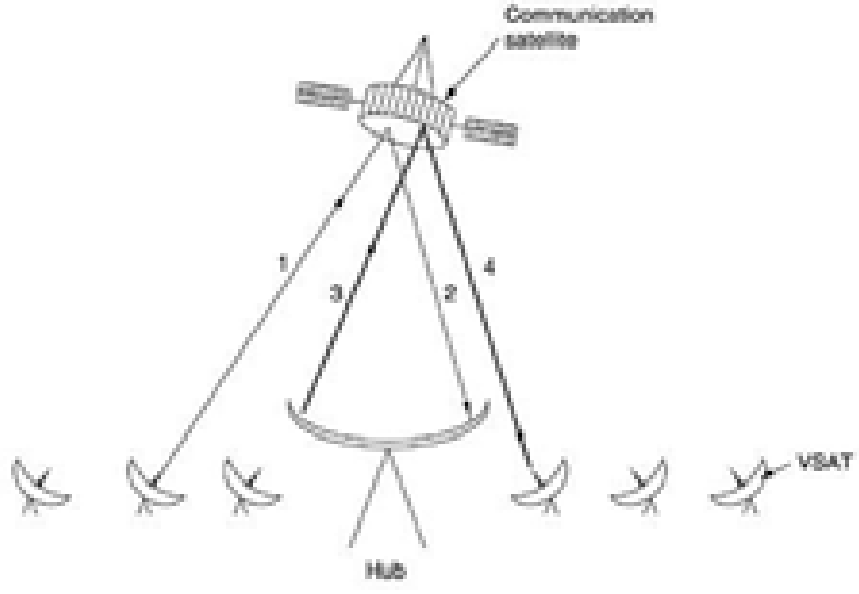


Figure 6-3. VSATs using a hub.

VSATs have great potential in rural areas. It is not widely appreciated, but over half the world's population lives over an hour's walk from the nearest telephone. Stringing telephone wires to thousands of small villages is far beyond the budgets of most.

Third World governments, but installing 1-meter VSAT dishes powered by solar cells is often feasible. VSATs provide the technology that will wire the world. Communication satellites have several properties that are radically different from terrestrial point-to-point links.

To begin with, even though signals to and from a satellite travel at the speed of light (nearly 300,000 km/sec), the long round-trip distance introduces a substantial delay for GEO satellites. Depending on the distance between the user and the ground station, and the elevation of the satellite above the horizon, the end-to-end transit time is between 250 and 300 msec.

A typical value is 270 msec (540 msec for a VSAT system with a hub). For comparison purposes, terrestrial microwave links have a propagation delay of roughly 3 μsec/km, and coaxial cable or fiber optic links have a delay of approximately 5 μsec/km. The latter is slower than the former because electromagnetic signals travel faster in air than in solid materials.

Another important property of satellites is that they are inherently broadcast media. It does not cost more to send a message to thousands of stations within a transponder's footprint than it does to send to one. For some

applications, this property is very useful. For example, one could imagine a satellite broadcasting popular Web pages to the caches of a large number of computers spread over a wide area.

Even when broadcasting can be simulated with point-to-point lines, satellite broadcasting may be much cheaper. On the other hand, from a security and privacy point of view, satellites are a complete disaster: everybody can hear everything. Encryption is essential when security is required.

Satellites also have the property that the cost of transmitting a message is independent of the distance traversed. A call across the ocean costs no more to service than a call across the street. Satellites also have excellent error rates and can be deployed almost instantly, a major consideration for military communication.

6.3 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.

6.4 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.

6.4.1 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.

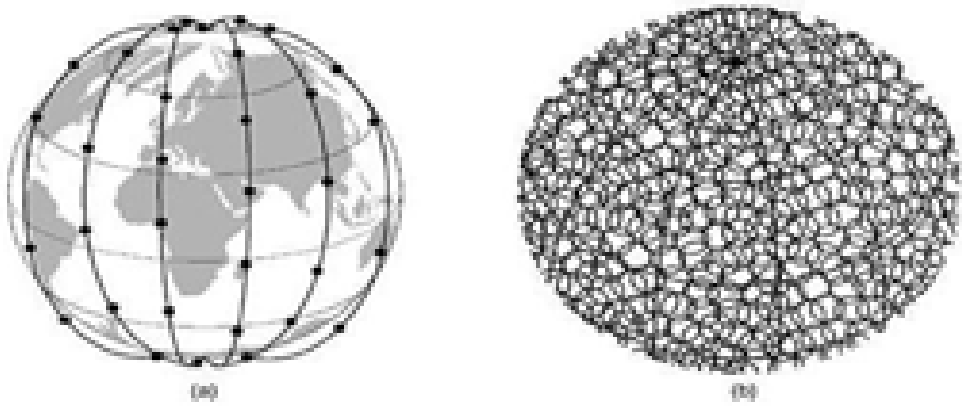
This proposal set off a feeding frenzy among other communication companies. All of a sudden, everyone wanted to launch a chain of low-orbit satellites. After seven years of cobbling together partners and financing, the partners launched the Iridium satellites in 1997.

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 6-4(a)People not knowing much about chemistry can think of this arrangement as a very, very big dysprosium atom, with the earth as the



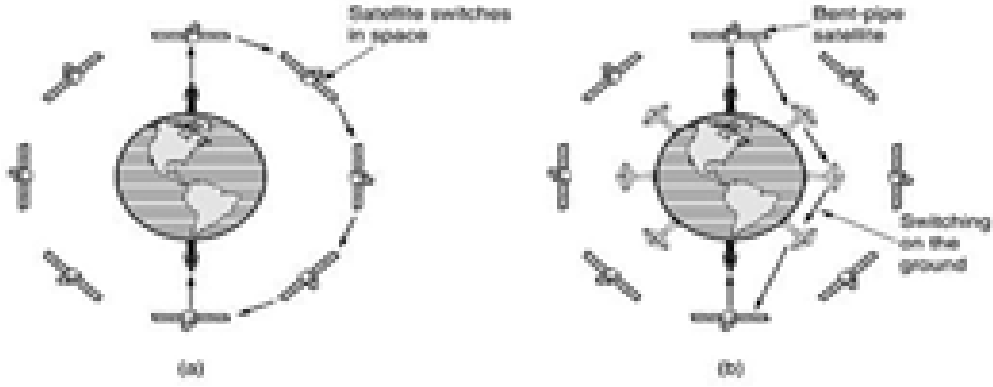
nucleus and the satellites as the electrons.

Figure 6-4. (a) The Iridium satellites form six necklaces around the earth. (b) 1628 moving cells cover the earth.

Each satellite has a maximum of 48 cells (spot beams), with a total of 1628 cells over the surface of the earth, as shown in Fig. 6-4(b). Each satellite has a capacity of 3840 channels, or 253,440 in all. Some of these are used for paging and navigation, while others are used for data and voice.

An interesting property of Iridium is that communication between distant customers takes place in space, with one satellite relaying data to the next one, as illustrated in Fig 6-5(a). Here we see a caller at the North Pole contacting a

satellite directly overhead. The call is relayed via other satellites and finally sent



down to the callee at the South Pole.

Figure 6-5. (a) Relaying in space. (b) Relaying on the ground.

6.4.2 Globalstar

An alternative design to Iridium is Globalstar. 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 in Fig 6-5(b) 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 mill watts of power, so the signal that gets back to the ground station is fairly weak, even after having been amplified by the satellite.

6.4.3 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 very slow speed 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.

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 un-crowded 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.

6.5 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 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 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 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 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 (e.g., some future space shuttle can toss out dozens of satellites on one launch) or low-orbit satellites catch on in a big way, it is not certain that fiber will win in all markets.

6.6 Summary

1. A communication satellite is a big microwave repeater in the sky which 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.
2. Geostationary satellites are satellite at an altitude of 35,800 km in a circular equatorial orbit which would appear to remain motionless in the sky.
3. Satellites are classified as medium orbit and low orbit with respect to their spacing from the earth.
4. The mainstream communication of the future will be terrestrial fiber optics combined with cellular radio, but for some specialized uses, satellites are better.

UNIT - III

Lesson 7

Error Detection and Correction

Contents

7.0 Aim

Computer Networks enable us to transfer data between systems in a very fast manner. However, due to some unexpected conditions, errors can occur. It is very important to detect that error has occurred during the data transfer and there is a need to correct them if possible. Lot of techniques is available to detect and correct errors that may occur during the data transfer. Learning about those techniques is the objective.

7.1 Introduction

The telephone system has three parts:

1. Switches
2. Interoffice trunks
3. Local loops

The first two parts are digital in most of the 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.

7.2 Burst Errors

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.

Advantage

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.

Disadvantage

They are much harder to correct than are isolated errors.

7.3 Error-Correcting Codes

Network designers have developed two basic strategies for dealing with errors.

- ✓ Include enough redundant information along with each block of data sent, to enable the receiver to deduce what the transmitted data must have been.
- ✓ 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.

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

To understand how errors can be handled, it is necessary to look closely at what an error really is.

- Normally, a frame consists of m data (i.e., message) bits and r redundant, or check, bits.
- Let the total length be n (i.e., n = m + r).
- An n-bit unit containing data and check bits is often referred to as an n-bit codeword.
- Given any two code-words, say, 10001001 and 10110001, it is possible to determine how many corresponding bits differ. In this case, 3 bits differ.
- To determine how many bits differ, just exclusive OR the two code-words and count the number of 1 bits in the result, for example:

```
  10001001
  10110001
  --------
  00111000
```

7.3.1 Hamming distance

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 the minimum. This distance is the Hamming distance of the complete code.

The error-detecting and error-correcting properties of a code depend on its Hamming distance. To detect d errors, you need a distance d + 1 code because with such a code there is no way that d single-bit errors can change a valid codeword into another valid codeword.

When the receiver sees an invalid codeword, it can tell that a transmission error has occurred. Similarly, to correct d errors, you need a distance 2d + 1 code because that way the legal code-words are so far apart that even with d changes, the original codeword is still closer than any other codeword, so it can be uniquely determined.

7.3.2 Example – Error Detection

Consider a code in which a single parity bit is appended to the data. The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd).

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

7.3.3 Example – Error Correction

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 codeword 0000000111 arrives, the receiver knows that the original must have been 0000011111. If, however, a triple error changes 0000000000 into 0000000111, the error will not be corrected properly.

Imagine that we want to design a code with m message bits and r check bits that will allow all single errors to be corrected. Each of the 2^m legal messages has n illegal code-words at a distance 1 from it.

These are formed by systematically inverting each of the n bits in the n-bit codeword formed from it. Thus, each of the 2^m legal messages requires n + 1 bit patterns dedicated to it. Since the total number of bit patterns is 2^n, we must have $(n + 1)2^m \leq 2^n$. 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 codeword 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 codeword 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 codeword 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. Figure 7--1 shows some 7-bit ASCII characters encoded as 11-bit codewords using a Hamming code. Remember that the data are found in bit positions 3, 5, 6, 7, 9, 10, and 11.

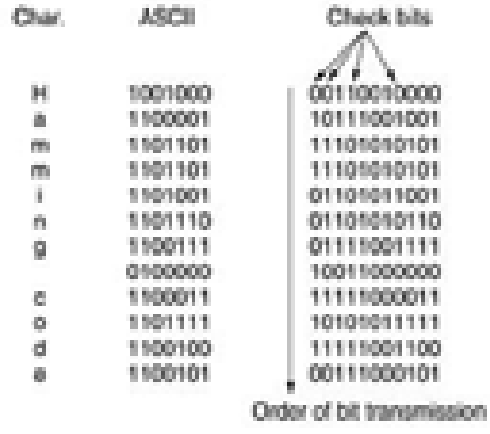


Figure 7--1. Use of a Hamming code to correct burst errors.

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 codewords are arranged as a matrix, one codeword per row. Normally, the data would be transmitted one codeword 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, and so on, as indicated in Fig. 7-1.

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 codewords will have been affected, but the Hamming code can correct one error per codeword, 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.

7.4 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.

7.4.1 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 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 n + 1 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}.

7.4.2 CRC (Cyclic Redundancy Check)

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:

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

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 checksummed frame is divisible by G(x). When the receiver gets the checksummed frame, it tries dividing it by G(x). If there is a remainder, there has been a transmission error.

The algorithm for computing the checksum is as follows:

1. Let r be the degree of G(x). Append r zero bits to the low-order end of the frame so it now contains m + r bits and corresponds to the polynomial $x^rM(x)$.

2. Divide the bit string corresponding to G(x) into the bit string corresponding to $x^rM(x)$, using modulo 2 division.

3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to $x^rM(x)$ using modulo 2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial T(x).

Figure 7--2 illustrates 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.

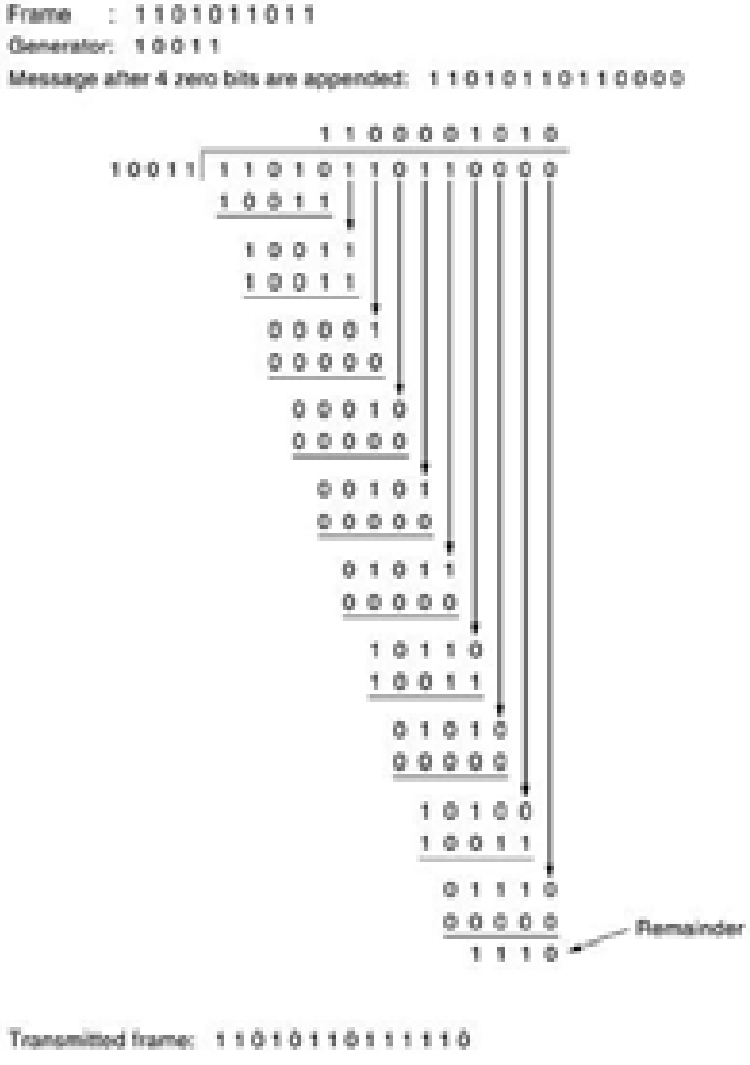


Figure 7--2. Calculation of the polynomial code checksum

7.4.3 Analysis of the power of this method

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 checksum 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 1 + 1 = 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 ≤ r. A burst error of length k can be represented by $x^i(x^{k-1} + \ldots + 1)$, 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 $\frac{1}{2}^{r-1}$. 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 $\frac{1}{2}^r$, assuming that all bit patterns are equally likely.

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

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$

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

7.5 Summary

- Burst Errors have both advantages and disadvantages.
- Hamming Code provides us a way to detect and correct errors.
- There are some methods which provide only error detection.
- Cyclic Redundancy Checks and Check Sums also play an important role in error detection and correction

Lesson 8

Elementary Data Link Protocols and Sliding Window Protocols

Contents

8.0 Aim

To study about the design principles for layer 2, the data link layer. This study deals with the algorithms for achieving reliable, efficient communication between two adjacent machines at the data link layer. By adjacent, we mean that the two machines are connected by a communication channel that acts conceptually like a wire (e.g., a coaxial cable, telephone line, or point-to-point wireless channel). The essential property of a channel that makes it "wirelike" is that the bits are delivered in exactly the same order in which they are sent. There are a lot of limitations which have implications for the efficiency of the data transfer. The protocols used for communications must take all these factors into consideration. These protocols are the subject of this chapter.

8.1 Introduction

Communication circuits make errors occasionally. Furthermore, they have only a finite data rate, and there is a nonzero propagation delay between the time a bit is sent and the time it is received. These limitations have important implications for the efficiency of the data transfer. The protocols used for communications must take all these factors into consideration. These protocols are the subject of this chapter.

8.2 Elementary Data Link Protocols

To introduce the subject of protocols, we will begin by looking at three protocols of increasing complexity. 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. 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 8-1 shows 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.

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.

```
#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
```

Figure 8-1. Some definitions needed in the protocols to follow. These definitions are located in the file protocol.h.

A number of procedures are also listed in Fig. 8-1. 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 Fig. 3-2-1 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.h, and using the #include facility of the C preprocessor to include them in the protocol files.

8.2.1 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,'' is shown in Figure 8-2.

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.

```
/* 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 */
  }
}
```

Figure 8-2. An unrestricted simplex protocol.

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. 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.

8.2.2 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 Δt to execute from_physical_layer plus to_network_layer, the sender must transmit at an average rate less than one frame per time Δt.

Moreover, if we assume that no automatic buffering and queueing are done within the receiver's hardware, the sender must never transmit a new frame until the old one has been fetched by from_physical_layer, lest the new one overwrite the old one.

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. Figure 8-3 gives an example of 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 */
  }
}
```

Figure 8-3. A simplex stop-and-wait protocol.

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.

8.3 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 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. The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permitted to send. These frames are said to fall within the sending window. Similarly, the receiver also maintains a receiving window corresponding to the set of frames it is permitted to accept.

The sender's window and the receiver's window need not have the same lower and upper limits or even have the same size. In some protocols they are fixed in size, but in others they can grow or shrink over the course of time as frames are sent and received. Although these protocols give the data link layer more freedom about the order in which it may send and receive frames, we have definitely not dropped the requirement that the protocol must deliver packets to the destination network layer in the same order they were passed to the data link layer on the sending machine.

Nor have we changed the requirement that the physical communication channel is "wire-like," that is, it must deliver all frames in the order sent. The sequence numbers within the sender's window represent frames that have been sent or can be sent but are as yet not acknowledged. Whenever a new packet arrives from the network layer, it is given the next highest sequence number, and the upper edge of the window is advanced by one. When an acknowledgement comes in, the lower edge is advanced by one. In this way the window continuously maintains a list of unacknowledged frames. Figure 8-4 shows an example.

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.

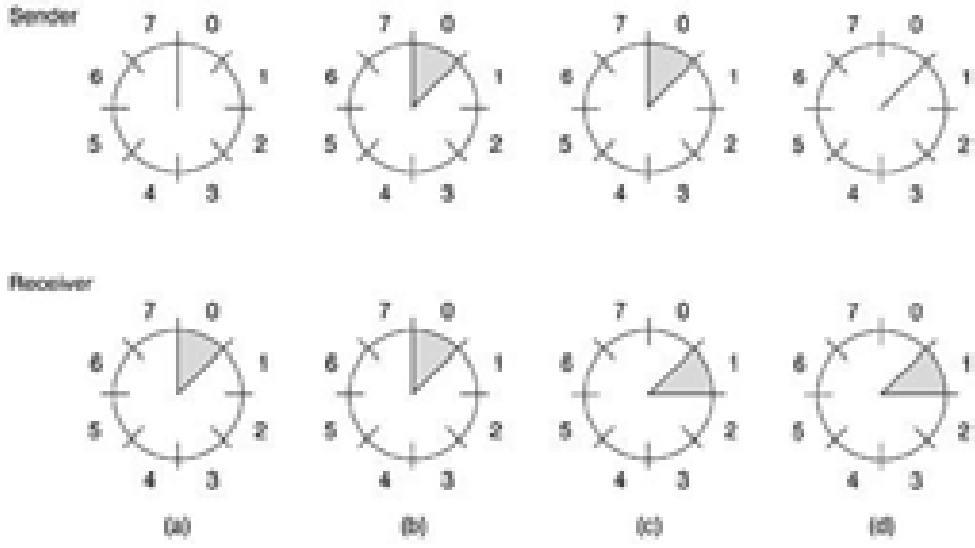


Figure 8-4. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

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. Figure 8-4 shows an example with a maximum window size of 1. Initially, no frames are outstanding, so the lower and upper edges of the sender's window are equal, but as time goes on, the situation progresses as shown.

8.3.1 A One-Bit Sliding Window Protocol

Before tackling the general case, let us first examine a sliding window protocol with a maximum window size of 1. Such a protocol uses stop-and-wait since the sender transmits a frame and waits for its acknowledgement before sending the next one. Figure 8-5 depicts such a protocol. Like the others, 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.

```
/* 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 */
  }
}
```

Figure 8-5. A 1-bit sliding window protocol.

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 buffer and can fetch the next packet from its network layer.

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. This synchronization difficulty is illustrated by Fig 8-6. 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.

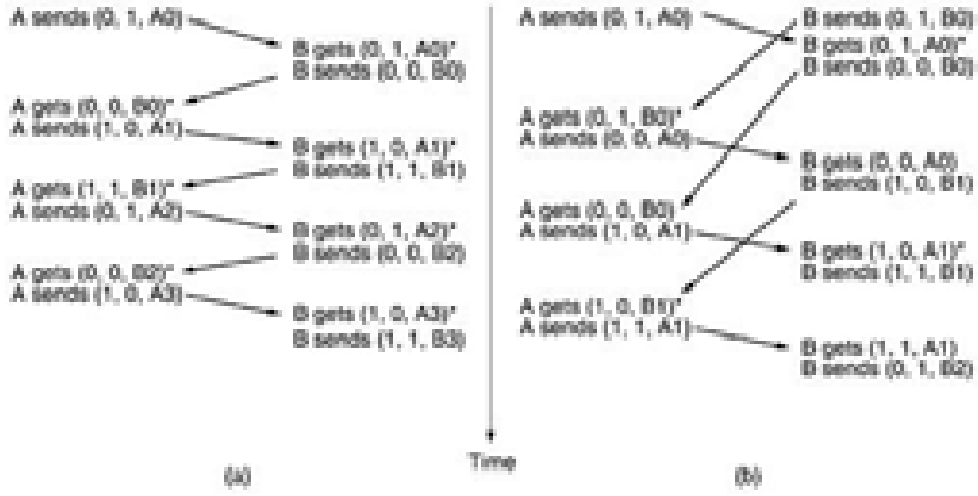


Figure 8-6. Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

8.3.2 A Protocol Using Go Back N

The usual assumption is 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

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.

In Fig 8-7 we see the effects of pipelining on error recovery. We will now examine it in some detail.

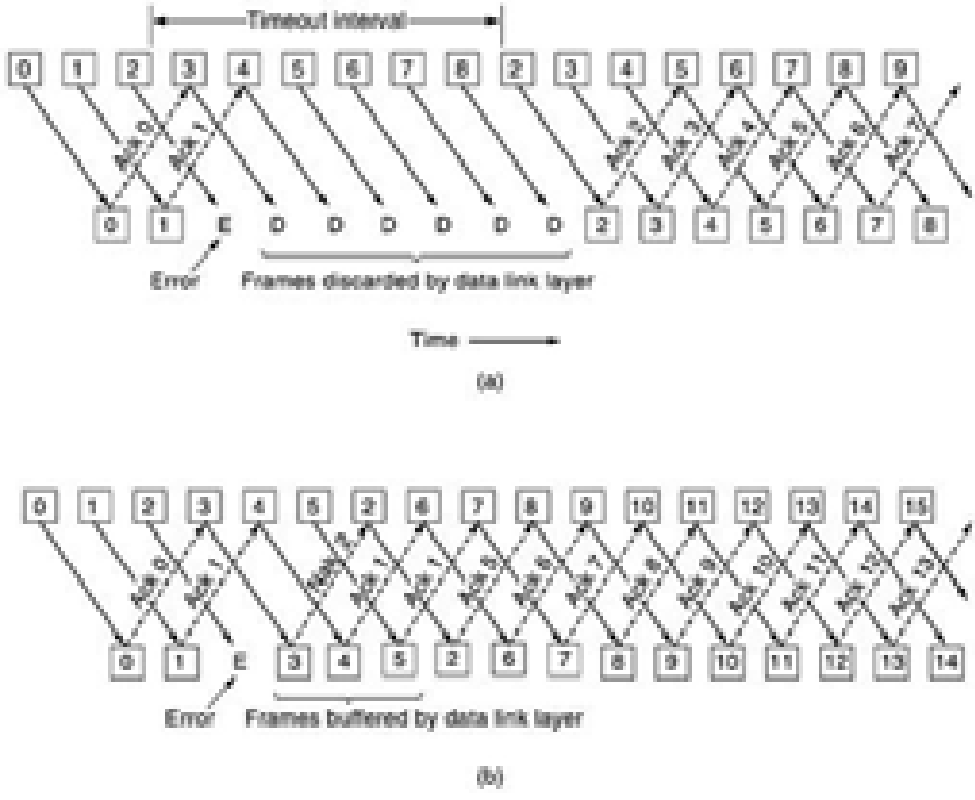


Figure 8-7. Pipelining and error recovery. Effect of an error when (a) receiver's window size is 1 and (b) receiver's window size is large.

Two basic approaches are available for dealing with errors in the presence of pipelining.

1. 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 8-7(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.

2. 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 is 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 Fig 8-7(b), 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. Figure 8-8 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.

```
/* Protocol 5 (go back n) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike in the previous
   protocols, the network layer is not assumed to have a new packet all the time. Instead,
   the network layer causes a network_layer_ready event when there is a packet to send. */

#define MAX_SEQ 7                       /* should be 2^n - 1 */
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if a <=b < c circularly; false otherwise. */
  if ((((a <= b) && (b < c)) || ((c < a) && (a <= b))) || (((b < c) && (c < a))))
      return(true);
    else
      return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Construct and send a data frame. */
  frame s;                              /* scratch variable */

  s.info = buffer[frame_nr];            /* insert packet into frame */
  s.seq = frame_nr;                     /* insert sequence number into frame */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);/* piggyback ack */
  to_physical_layer(&s);                /* transmit the frame */
  start_timer(frame_nr);                /* start the timer running */
}

void protocol5(void)
{
  seq_nr next_frame_to_send;            /* MAX_SEQ > 1; used for outbound stream */
  seq_nr ack_expected;                  /* oldest frame as yet unacknowledged */
  seq_nr frame_expected;                /* next frame expected on inbound stream */
  frame r;                              /* scratch variable */
  packet buffer[MAX_SEQ + 1];           /* buffers for the outbound stream */
  seq_nr nbuffered;                     /* # output buffers currently in use */
  seq_nr i;                             /* used to index into the buffer array */
  event_type event;

  enable_network_layer();               /* allow network_layer_ready events */
  ack_expected = 0;                     /* next ack expected inbound */
  next_frame_to_send = 0;               /* next frame going out */
  frame_expected = 0;                   /* number of frame expected inbound */
  nbuffered = 0;                        /* initially no packets are buffered */

  while (true) {
    wait_for_event(&event);             /* four possibilities: see event_type above */
```

```
switch(event) {
  case network_layer_ready:          /* the network layer has a packet to send */
       /* Accept, save, and transmit a new frame. */
       from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
       nbuffered = nbuffered + 1;    /* expand the sender's window */
       send_data(next_frame_to_send, frame_expected, buffer);/* transmit the frame */
       inc(next_frame_to_send);      /* advance sender's upper window edge */
       break;

  case frame_arrival:                /* a data or control frame has arrived */
       from_physical_layer(&r);      /* get incoming frame from physical layer */

       if (r.seq == frame_expected) {
            /* Frames are accepted only in order. */
            to_network_layer(&r.info); /* pass packet to network layer */
            inc(frame_expected);     /* advance lower edge of receiver's window */
       }

       /* Ack n implies n - 1, n - 2, etc. Check for this. */
       while (between(ack_expected, r.ack, next_frame_to_send)) {
            /* Handle piggybacked ack. */
            nbuffered = nbuffered - 1; /* one frame fewer buffered */
            stop_timer(ack_expected); /* frame arrived intact; stop timer */
            inc(ack_expected);       /* contract sender's window */
       }
       break;

  case cksum_err: break;             /* just ignore bad frames */

  case timeout:                      /* trouble; retransmit all outstanding frames */
       next_frame_to_send = ack_expected;   /* start retransmitting here */
       for (i = 1; i <= nbuffered; i++) {
            send_data(next_frame_to_send, frame_expected, buffer);/* resend frame */
            inc(next_frame_to_send);  /* prepare to send the next one */
       }

  }

  if (nbuffered < MAX_SEQ)
       enable_network_layer();
  else
       disable_network_layer();
}
}
```

Figure 8-8. A sliding window protocol using go back n.

- 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.
    1. The sender sends frames 0 through 7.
    2. A piggybacked acknowledgement for frame 7 eventually comes back to the sender.

3. The sender sends another eight frames, again with sequence numbers 0 through 7.
4. 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.
- As an illustration of how the timers could be implemented, consider the example of Fig 8-9(a).
  - ✓ 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 8-9(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.

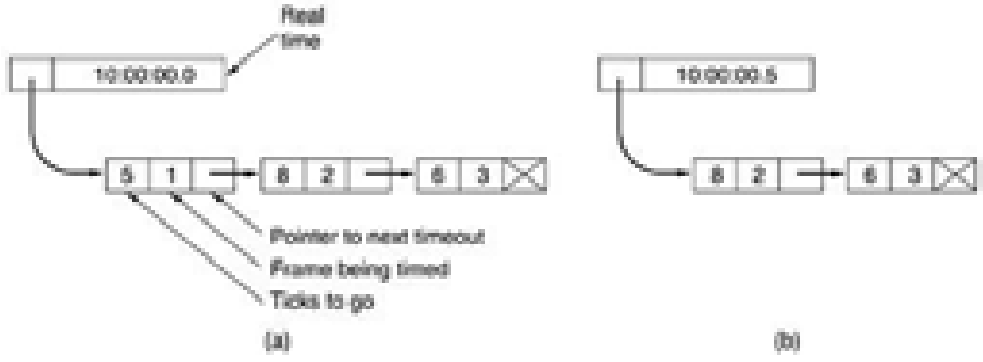


Figure 8-9. Simulation of multiple timers in software.

8.4 Summary

- The task of the data link layer is to convert the raw bit stream offered by the physical layer into a stream of frames for use by the network layer.
- Various framing methods are used, including character count, byte stuffing, and bit stuffing. Data link protocols can provide error control to retransmit damaged or lost frames.
- To prevent a fast sender from overrunning a slow receiver, the data link protocol can also provide flow control. The sliding window mechanism is widely used to integrate error control and flow control in a convenient way.
- Sliding window protocols can be categorized by the size of the sender's window and the size of the receiver's window.
- When both are equal to 1, the protocol is stop-and-wait.
- When the sender's window is greater than 1, for example, to prevent the sender from blocking on a circuit with a long propagation delay, the receiver can be programmed either to discard all frames other than the next one in sequence or to buffer out-of-order frames until they are needed.

Lesson 9

Multiple Access Protocols

Contents

9.0 Aim

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.

9.1 Introduction

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.

9.2 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.

9.2.1 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. A sketch of frame generation in an ALOHA system is given in Fig. 9-1.

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.

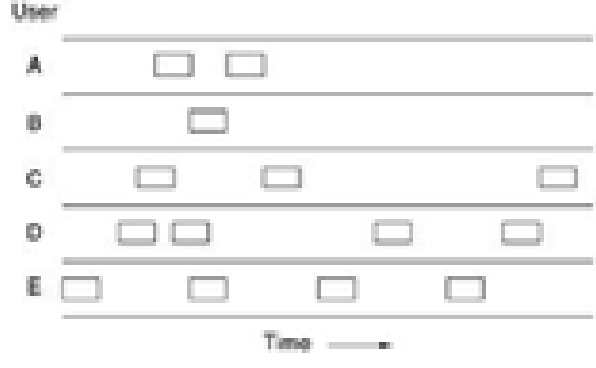


Figure 9-1. In pure ALOHA, frames are transmitted at completely arbitrary times.

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 all loads, 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.

A frame will not suffer a collision if no other frames are sent within one frame time of its start, as shown in Fig. 9-2. 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.

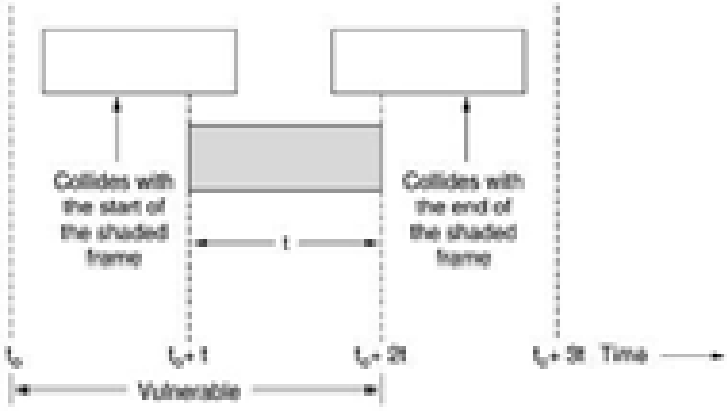


Figure 9-2. Vulnerable period for the shaded frame

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

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

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. 9-3. 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.

9.2.2 Slotted ALOHA

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

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

As you can see from Fig. 9-3, 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 Eq. 4-2).

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

$$P_k = e^{-G}(1 - e^{-G})^{k-1}$$

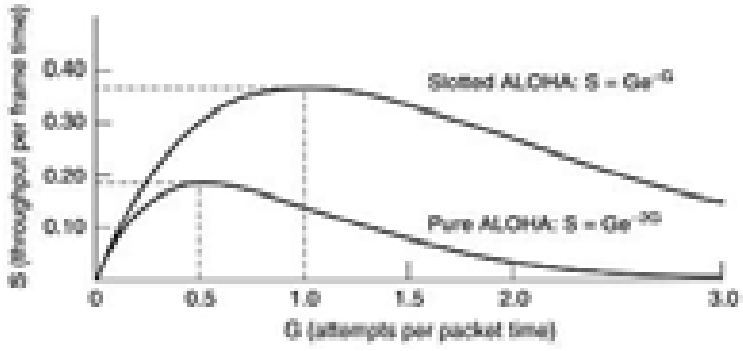


Figure 9-3. Throughput versus offered traffic for ALOHA systems.

The expected number of transmissions, E, per carriage return typed is then

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

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.

9.3 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.

9.3.1 Persistent and Nonpersistent 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. Figure 9-4 shows the computed throughput versus offered traffic for all three protocols, as well as for pure and slotted ALOHA.

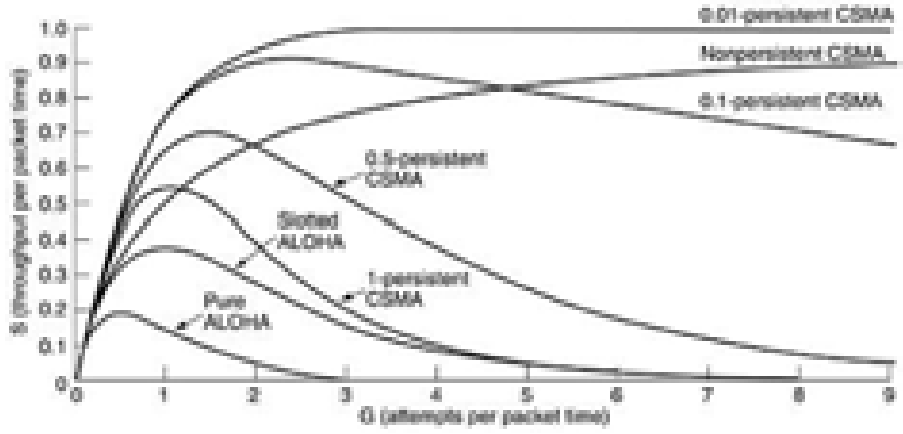


Figure 9-4. Comparison of the channel utilization versus load for various random access protocols

9.3.2 CSMA with Collision Detection

Persistent and nonpersistent CSMA protocols are clearly an improvement over ALOHA because they ensure that no station begins to transmit when it senses the channel busy. Another improvement is for stations to abort their transmissions as soon as they detect a collision.

In other words, if two stations sense the channel to be idle and begin transmitting simultaneously, they will both detect the collision almost immediately. Rather than finish transmitting their frames, which are irretrievably garbled anyway, they should abruptly stop transmitting as soon as the collision is detected.

Quickly terminating damaged frames saves time and bandwidth. This protocol, known as CSMA/CD (CSMA with Collision Detection) is widely used on LANs in the MAC 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 of Fig. 9-5.

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

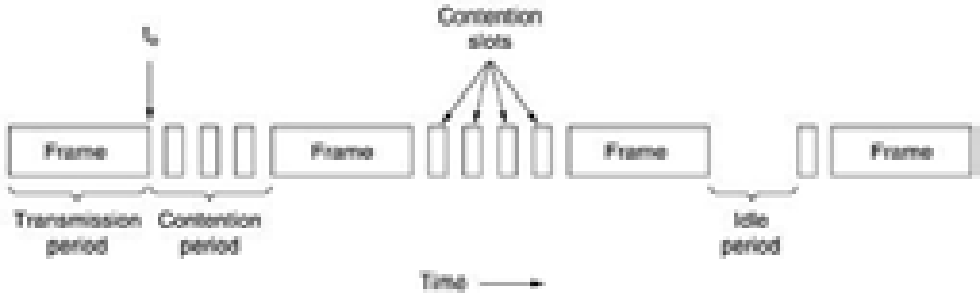


Figure 9-5. CSMA/CD can be in one of three states: contention, transmission, or idle

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 an 1 km long coaxial cable, $\tau \approx 5$ 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-sublayer 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).

9.4 Collision-Free Protocols

Although collisions do not occur with CSMA/CD once a station has unambiguously captured the channel, they can still occur during the contention period. These collisions adversely affect the system performance, especially when the cable is long (i.e., large τ) and the frames are short.

And CSMA/CD is not universally applicable. In this section, we will examine some protocols that resolve the contention for the channel without any collisions at all, not even during the contention period. Most of these are not currently used in major systems, but in a rapidly changing field, having some protocols with excellent properties available for future systems is often a good thing.

In the protocols to be described, we assume that there are exactly N stations, each with a unique address from 0 to N - 1 "wired" into it. It does not matter that some stations may be inactive part of the time. We also assume that propagation delay is negligible. The basic question remains: Which station gets the channel after a successful transmission? We continue using the model of Fig. 9-5 with its discrete contention slots.

9.4.1 A Bit-Map Protocol

In our first collision-free protocol, the basic bit-map method, each contention period consists of exactly N slots. If station 0 has a frame to send, it transmits a 1 bit during the zeroth slot. No other station is allowed to transmit during this slot. Regardless of what station 0 does, station 1 gets the opportunity to transmit a 1 during slot 1, but only if it has a frame queued.

In general, station j may announce that it has a frame to send by inserting a 1 bit into slot j. After all N slots have passed by, each station has

complete knowledge of which stations wish to transmit. At that point, they begin transmitting in numerical order (see Fig. 9-6).

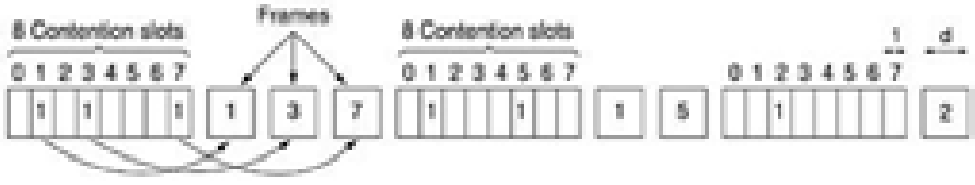


Figure 9-6. The basic bit-map protocol

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.

9.4.2 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. 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 1. 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. 9-7. 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.

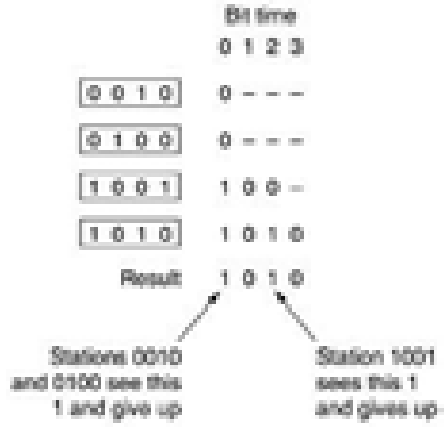


Figure 9-7. The binary countdown protocol. A dash indicates silence.

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.

9.5 Limited-Contention Protocols

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. Substituting p = 1/k, we get

$$\Pr[success\ with\ optimal\ p] = \left[\frac{k-1}{k}\right]^{k=1}$$

This probability is plotted in Fig. 9-8. 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.

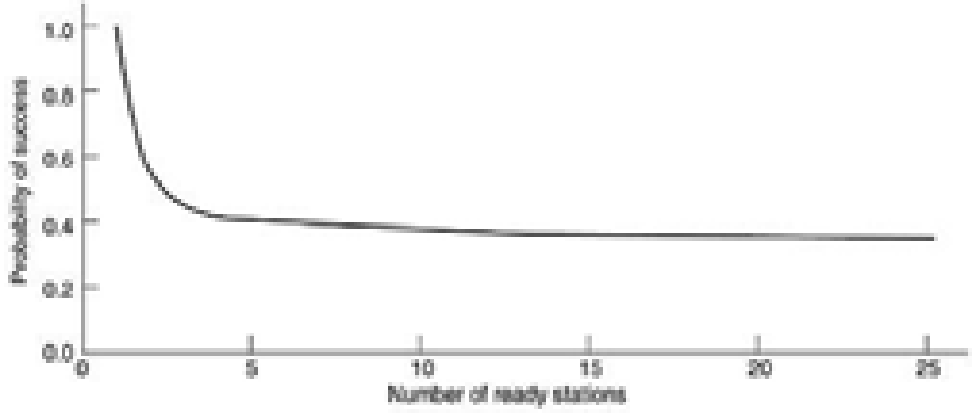


Figure 9-8. Acquisition probability for a symmetric contention channel.

From Fig. 9-8, 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 Fig. 9-8. 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.

9.5.1 The Adaptive Tree Walk Protocol

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, as illustrated in Fig. 9-9. 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.

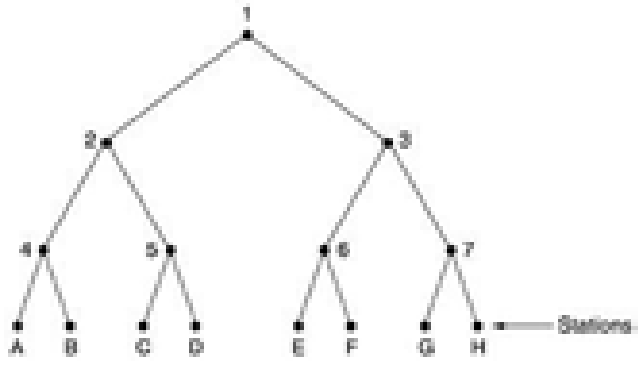


Figure 9-9. The tree for eight stations.

In essence, if a collision occurs during slot 0, the entire tree is searched, depth first, to locate all ready 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.

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 in Fig. 9-9 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 $i = \log_2 q$.

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. 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.

9.6 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.

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 sublayer protocols.

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.

Unlike cellular telephone systems, each cell has only one channel, covering the entire available bandwidth and covering all the stations in its cell. Typically, its bandwidth is 11 to 54 Mbps. 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. 9-10, 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. 9-10(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.

Figure 9-10. A wireless LAN. (a) A transmitting. (b) B transmitting.

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. 9-10(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.

9.6.1 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. 9-11.

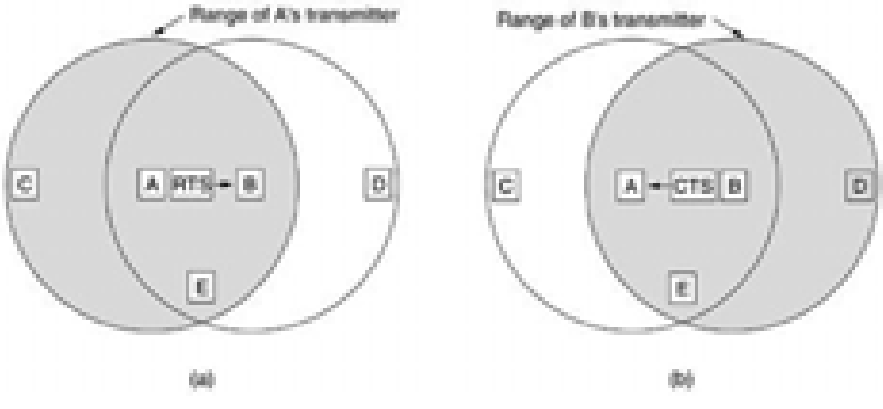


Figure 9-11. The MACA protocol. (a) A sending an RTS to B. (b) B responding with a CTS to A

9.7 Summary

- The simplest allocation schemes are FDM and TDM. These are efficient when the number of stations is small and fixed and the traffic is continuous. Both are widely used under these circumstances, for example, for dividing up the bandwidth on telephone trunks.
- When the number of stations is large and variable or the traffic is fairly bursty, FDM and TDM are poor choices.

- The ALOHA protocol, with and without slotting, has been proposed as an alternative.
- ALOHA and its many variants and derivatives have been widely discussed, analyzed, and used in real systems.
- Binary countdown completely eliminates contention.
- The biggest problem in Wireless LANs is caused by hidden stations, so CSMA does not work. One class of solutions, typified by MACA and MACAW, attempts to stimulate transmissions around the destination, to make CSMA work better.
- Frequency hopping spread spectrum and direct sequence spread spectrum are also used. IEEE 802.11 combines CSMA and MACAW to produce CSMA/CA.

Lesson 10

Ethernet

Contents

10.0 Aim

To learn about the Ethernet and various concepts regarding it. Ethernet is an IEEE 802.3 standard for contention networks. Ethernet uses a bus or star topology and relies on the form of access known as Carrier Sense Multiple Access with Collision Detection (CSMA/DC) to regulate communication line traffic. Network nodes are linked by coaxial cable, fiber-optic cable, or by twisted-pair wiring. Data is transmitted in variable-length frames containing delivery and control information and up to 1,500 bytes of data. The Ethernet standard provides for base-band transmission at 10 megabits (10 million bits) per second.

10.1 Introduction

The IEEE has standardized a number of local area networks and metropolitan area networks under the name of IEEE 802. A few have survived but many have not. 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. Both 802.3 and 802.11 have different physical layers and

different MAC sublayers but converge on the same logical link control sublayer (defined in 802.2), so they have the same interface to the network layer.

10.2 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, as shown in Figure 10-1.

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

Figure 10-1. The most common kinds of Ethernet cabling

10.2.1 10Base5 Cabling

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.

10.2.2 10Base2 Cabling

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.

10.2.3 10Base-T Cabling

Usually, these wires are telephone company twisted pairs, since most office buildings are already wired this way, and normally plenty of spare pairs are available. This scheme is called 10Base-T.

These three wiring schemes are illustrated in Figure 10-2.

10.2.4 10Base-F Cabling

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.

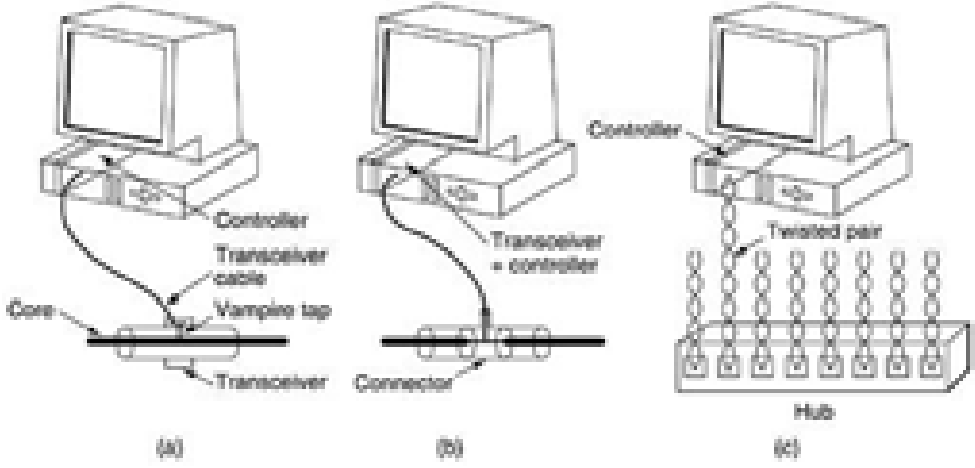


Figure 10-2. Three kinds of Ethernet cabling (a) 10Base5. (b) 10Base2. (c) 10Base-T.

Figure 10-3 shows different ways of wiring a building. In Fig. 10-3(a), a single cable is snaked from room to room, with each station tapping into it at the nearest point. In Fig. 10-3(b), a vertical spine runs from the basement to the roof, with horizontal cables on each floor connected to the spine by special amplifiers (repeaters).

In some buildings, the horizontal cables are thin and the backbone is thick. The most general topology is the tree, as in Fig. 10-3(c), because a network with two paths between some pairs of stations would suffer from interference between the two signals.

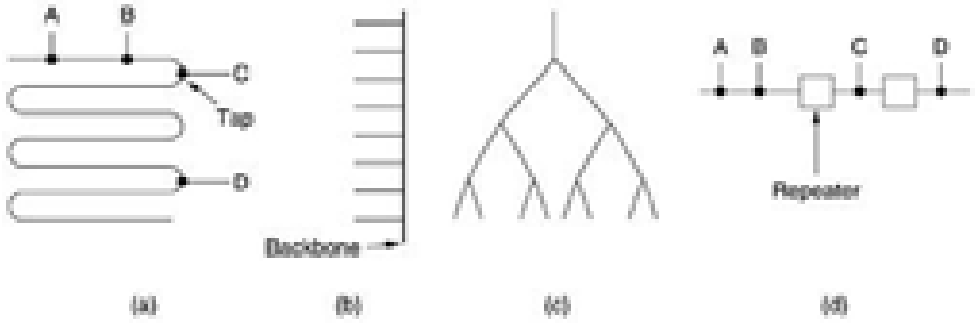


Figure 10-3. Cable topologies. (a) Linear. (b) Spine. (c) Tree. (d) Segmented.

Each version of Ethernet has a maximum cable length per segment. To allow larger networks, multiple cables can be connected by repeaters, as shown in Fig. 10-3(d).

A repeater is a physical layer device. It receives, amplifies (regenerates), and retransmits signals in both directions. As far as the software is concerned, a series of cable segments connected by repeaters is no different from a single cable (except for some delay introduced by the repeaters).

A system may contain multiple cable segments and multiple repeaters, but no two transceivers may be more than 2.5 km apart and no path between any two transceivers may traverse more than four repeaters.

10.3 Manchester Encoding

Two approaches called Manchester encoding and Differential Manchester encoding exist.

10.3.1 Manchester Encoding

With Manchester encoding, each bit period is divided into two equal intervals. A binary 1 bit is sent by having the voltage set high during the first interval and low in the second one. A binary 0 is just the reverse: first low and then high. This scheme ensures that every bit period has a transition in the middle, making it easy for the receiver to synchronize with the sender.

A disadvantage of Manchester encoding is that it requires twice as much bandwidth as straight binary encoding because the pulses are half the width. For example, to send data at 10 Mbps, the signal has to change 20 million times/sec. Manchester encoding is shown in Fig. 10-4(b).

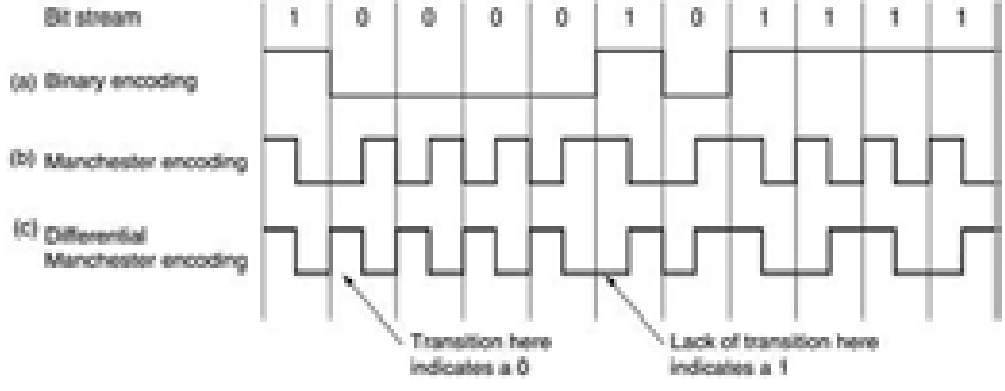


Figure 10-4. (a) Binary encoding. (b) Manchester encoding. (c) Differential Manchester encoding.

10.3.2 Differential Manchester encoding

Differential Manchester encoding, shown in Fig. 10-4(c), is a variation of basic Manchester encoding. In it, a 1 bit is indicated by the absence of a transition at the start of the interval. A 0 bit is indicated by the presence of a transition at the start of the interval. In both cases, there is a transition in the middle as well.

The differential scheme requires more complex equipment but offers better noise immunity. All Ethernet systems use Manchester encoding due to its simplicity. The high signal is + 0.85 volts and the low signal is - 0.85 volts, giving a DC value of 0 volts. Ethernet does not use differential Manchester encoding, but other LANs (e.g., the 802.5 token ring) do use it.

10.4 The Ethernet MAC Sublayer Protocol

The original DIX (DEC, Intel, Xerox) frame structure is shown in Fig. 10-5(a). 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.

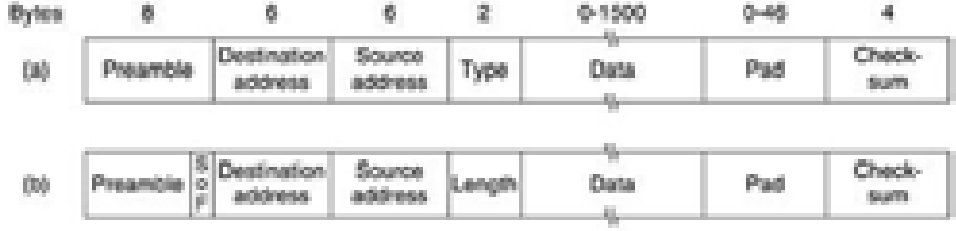


Figure 10-5. Frame formats. (a) DIX Ethernet. (b) IEEE 802.3.

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.

Another interesting feature of the addressing is the use of bit 46 (adjacent to the high-order bit) to distinguish local from global addresses. Local addresses are assigned by each network administrator and have no significance outside the local network.

Global addresses, in contrast, are assigned centrally by IEEE to ensure that no two stations anywhere in the world have the same global address. With 48 - 2 = 46 bits available, there are about 7×10^{13} global addresses. The idea is that any station can uniquely address any other station by just giving the right 48-bit number. It is up to the network layer to figure out how to locate the destination.

Next comes the Type field, which tells the receiver what to do with the frame. Multiple network-layer protocols may be in use at the same time on the same machine, so when an Ethernet frame arrives, the kernel has to know which one to hand the frame to. The Type field specifies which process to give the frame to.

Next comes the data, up to 1500 bytes. This limit was chosen somewhat arbitrarily at the time the DIX standard was cast in stone, mostly based on the fact that a transceiver needs enough RAM to hold an entire frame and RAM was expensive in 1978. A larger upper limit would have meant more RAM, hence a more expensive transceiver.

In addition to there being a maximum frame length, there is also a minimum frame length. While a data field of 0 bytes is sometimes useful, it causes a problem. When a transceiver detects a collision, it truncates the current frame, which means that stray bits and pieces of frames appear on the cable all the time.

To make it easier to distinguish valid frames from garbage, Ethernet requires that valid frames must be at least 64 bytes long, from destination address to checksum, including both. If the data portion of a frame is less than 46 bytes, the Pad field is used to fill out the frame to the minimum size.

Another (and more important) reason for having a minimum length frame is to prevent a station from completing the transmission of a short frame before the first bit has even reached the far end of the cable, where it may collide with another frame. The final Ethernet field is the Checksum. It is effectively a 32-bit hash code of the data.

If some data bits are erroneously received (due to noise on the cable), the checksum will almost certainly be wrong and the error will be detected. The checksum algorithm is a cyclic redundancy check (CRC). It just does error detection, not forward error correction. When IEEE standardized Ethernet, the committee made two changes to the DIX format, as shown in Fig. 10-5(b).

The first one was to reduce the preamble to 7 bytes and use the last byte for a Start of Frame delimiter, for compatibility with 802.4 and 802.5. The second one was to change the Type field into a Length field. Of course, now there was no way for the receiver to figure out what to do with an incoming frame, but that problem was handled by the addition of a small header to the data portion itself to provide this information.

Unfortunately, by the time 802.3 was published, so much hardware and software for DIX Ethernet was already in use that few manufacturers and users were enthusiastic about converting the Type field into a Length field. In 1997 IEEE threw in the towel and said that both ways were fine with it. Fortunately, all the Type fields in use before 1997 were greater than 1500.

Consequently, any number there less than or equal to 1500 can be interpreted as Length, and any number greater than 1500 can be interpreted as Type. Now IEEE can maintain that everyone is using its standard and everybody else can keep on doing what they were already doing without feeling guilty about it.

10.5 The Binary Exponential Back-off Algorithm

Let us now see how randomization is done when a collision occurs.

The model is that of Fig. 4-5. After a collision, time is divided into discrete slots whose length is equal to the worst-case round-trip propagation time on the ether (2□). 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 back-off, 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 back-off at 1023 keeps the bound from growing too large.

As described so far, CSMA/CD provides no acknowledgements. Since the mere absence of collisions does not guarantee that bits were not garbled by noise spikes on the cable, for reliable communication the destination must verify the checksum, and if correct, send back an acknowledgement frame to the source.

Normally, this acknowledgement would be just another frame as far as the protocol is concerned and would have to fight for channel time just like a data frame. However, a simple modification to the contention algorithm would allow speedy confirmation of frame receipt. All that would be needed is to reserve the first contention slot following successful transmission for the destination station. Unfortunately, the standard does not provide for this possibility.

10.6 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 back-off 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

$$A = kp(1-p)^{k-1}$$

A is maximized when p = 1/k, with A →1/e as 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

Since each slot has a duration 2□, the mean contention interval, w, is 2□/A. Assuming optimal p, the mean number of contention slots is never more than e, so w is at most 2□e ≈ 5.4□.

If the mean frame takes P sec to transmit, when many stations have frames to send,

$$\sum_{j=0}^{\infty} jA(1-A)^{j-1} = \frac{1}{A}$$

Here we see where the maximum cable distance between any two stations enters into the performance figures, giving rise to topologies other than that of Fig. 10-3(a). The longer the cable, the longer the contention interval. This observation is why the Ethernet standard specifies a maximum cable length.

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

$$Channel\ Efficiency = \frac{P}{P + 2\tau/A}$$

When the second term in the denominator is large, network efficiency will be low. More specifically, increasing network bandwidth or distance (the BL product) reduces efficiency for a given frame size.

Unfortunately, much research on network hardware is aimed precisely at increasing this product. People want high bandwidth over long distances (fiber optic MANs, for example), which suggests that Ethernet implemented in this manner may not be the best system for these applications.

In Fig. 10-7, the channel efficiency is plotted versus number of ready stations for 2□=51.2 μsec and a data rate of 10 Mbps, using Eq. (4-7). With a 64-byte slot time, it is not surprising that 64-byte frames are not efficient.

On the other hand, with 1024-byte frames and an asymptotic value of e 64-byte slots per contention interval, the contention period is 174 bytes long and the efficiency is 0.85.

To determine the mean number of stations ready to transmit under conditions of high load, we can use the following (crude) observation. Each frame ties up the channel for one contention period and one frame transmission time, for a total of P + w sec. The number of frames per second is therefore 1/(P + w). If each station generates frames at a mean rate of □ frames/sec, then when the system is in state k, the total input rate of all unblocked stations combined is k□ frames/sec. Since in equilibrium the input and output rates must be identical, we can equate these two expressions and solve for k. (Notice that w is a function of k.)

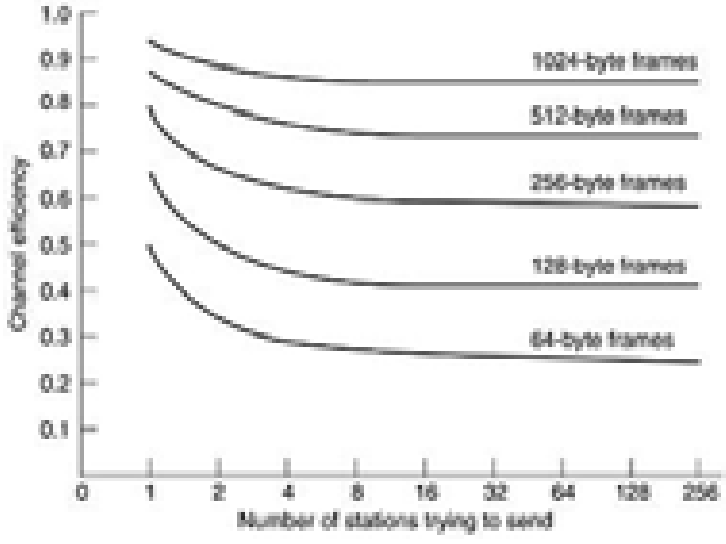


Figure 10-7. Efficiency of Ethernet at 10 Mbps with 512-bit slot times.

10.7 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, as shown in Fig. 10-8. 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.

When a station wants to transmit an Ethernet frame, it outputs a standard frame to the switch. The plug-in card getting the frame may check to see if it is destined for one of the other stations connected to the same card. If so, the frame is copied there. If not, the frame is sent over the high-speed backplane to the destination station's card. The backplane typically runs at many Gbps, using a proprietary protocol.

If two machines attached to the same plug-in card transmit frames at the same time, then: It depends on how the card has been constructed. One possibility is for all the ports on the card to be wired together to form a local on-card LAN.

Collisions on this on-card LAN will be detected and handled the same as any other collisions on a CSMA/CD network—with retransmissions using the binary exponential back-off algorithm.

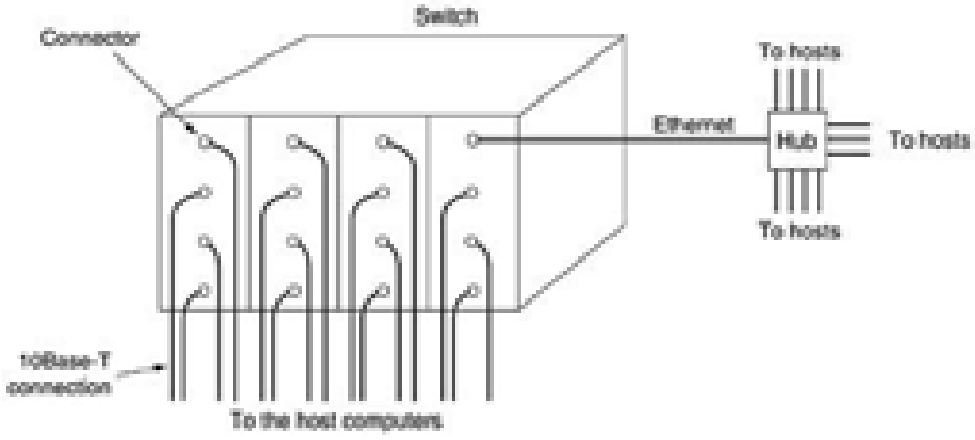


Figure 10-8. A simple example of switched Ethernet.

With this kind of plug-in card, only one transmission per card is possible at any instant, but all the cards can be transmitting in parallel. With this design, each card forms its own collision domain, independent of the others. With only one station per collision domain, collisions are impossible and performance is improved.

With the other kind of plug-in card, each input port is buffered, so incoming frames are stored in the card's on-board RAM as they arrive. This design allows all input ports to receive (and transmit) frames at the same time, for parallel, full-duplex operation, something not possible with CSMA/CD on a single channel. Once a frame has been completely received, the card can then check to see if the frame is destined for another port on the same card or for a distant port.

In the former case, it can be transmitted directly to the destination. In the latter case, it must be transmitted over the backplane to the proper card. With this design, each port is a separate collision domain, so collisions do not occur. The total system throughput can often be increased by an order of magnitude over 10Base5, which has a single collision domain for the entire system.

Since the switch just expects standard Ethernet frames on each input port, it is possible to use some of the ports as concentrators. In Fig. 10-8, the port in the upper-right corner is connected not to a single station, but to a 12-port hub. As frames arrive at the hub, they contend for the ether in the usual way, including collisions and binary backoff. Successful frames make it to the switch and are treated there like any other incoming frames: they are switched to the correct output line over the high-speed backplane. Hubs are cheaper than switches, but due to falling switch prices, they are rapidly becoming obsolete. Nevertheless, legacy hubs still exist.

10.8 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. 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.

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.

It was in this environment that IEEE reconvened the 802.3 committee in 1992 with instructions to come up with a faster LAN. One proposal was to keep 802.3 exactly as it was, but just make it go faster. Another proposal was to redo it totally to give it lots of new features, such as real-time traffic and digitized voice, but just keep the old name (for marketing reasons).

After some wrangling, the committee decided to keep 802.3 the way it was, but just make it go faster. The people behind the losing proposal did what any computer-industry people would have done under these circumstances—they stomped off and formed their own committee and standardized their LAN anyway (eventually as 802.12). It flopped miserably.

The 802.3 committee decided to go with a souped-up Ethernet for three primary reasons:

1. The need to be backward compatible with existing Ethernet LANs.
2. The fear that a new protocol might have unforeseen problems.
3. The desire to get the job done before the technology changed.

The work was done quickly (by standards committees' norms), and the result, 802.3u, was officially approved by IEEE in June 1995. Technically, 802.3u is not a new standard, but an addendum to the existing 802.3 standard (to emphasize its backward compatibility). Since practically everyone calls it fast Ethernet, rather than 802.3u, we will do that, too.

10.9 Wire Types

- Wire types are as shown in Fig. 10-9,

Name	Cable	Max. segment	Advantages
100Base-T4	Twisted pair	100 m	Uses category 3 UTP
100Base-TX	Twisted pair	100 m	Full duplex at 100 Mbps (Cat 5 UTP)
100Base-FX	Fiber optics	2000 m	Full duplex at 100 Mbps; long runs

Figure 10-9. The original fast Ethernet cabling.

10.10 Interconnection Devices

Two kinds of interconnection devices are possible with 100Base-T: hubs and switches, as shown in Fig. 10-8.

10.10.1 Hub

- ✓ In a hub, all the incoming lines (or at least all the lines arriving at one plug-in card) are logically connected, forming a single collision domain.

- ✓ All the standard rules, including the binary exponential backoff algorithm, apply, so the system works just like old-fashioned Ethernet.

10.10.2 Switch

- ✓ In a switch, each incoming frame is buffered on a plug-in line card and passed over a high-speed backplane from the source card to the destination card if need be.

10.11 IEEE 802.2: Logical Link Control

All that Ethernet and the other 802 protocols offer is a best-efforts datagram service. Sometimes, this service is adequate. For example, for transporting IP packets, no guarantees are required or even expected. An IP packet can just be inserted into an 802 payload field and sent on its way. If it gets lost, so be it.

There are also systems in which an error-controlled, flow-controlled data link protocol is desired. IEEE has defined one that can run on top of Ethernet and the other 802 protocols. In addition, this protocol, called LLC (Logical Link Control), hides the differences between the various kinds of 802 networks by providing a single format and interface to the network layer. LLC forms the upper half of the data link layer, with the MAC sublayer below it, as shown in Fig. 10-9.

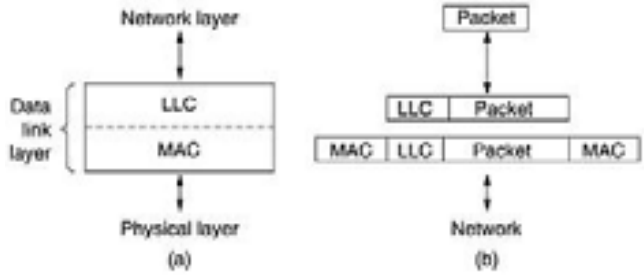


Figure 10-9. (a) Position of LLC. (b) Protocol formats.

Typical usage of LLC is as follocws.

The network layer on the sending machine passes a packet to LLC, using the LLC access primitives. The LLC sublayer then adds an LLC header, containing sequence and acknowledgement numbers. The resulting structure is then inserted into the payload field of an 802 frame and transmitted. At the receiver, the reverse process takes place.

LLC provides three service options: unreliable datagram service, acknowledged datagram service, and reliable connection-oriented service. The LLC header contains three fields: a destination access point, a source access point, and a control field. The access points tell which process the frame came from and where it is to be delivered, replacing the DIX Type field.

The control field contains sequence and acknowledgement numbers, very much in the style of HDLC (see Fig. 3-24), but not identical to it. These fields are primarily used when a reliable connection is needed at the data link level.

In the case of Internet, attempt to deliver IP the packets is only sufficient, so no acknowledgements at the LLC level are required.

10.12 Summary

- Ethernet is the dominant form of local area networking.
- It uses CSMA/CD for channel allocation.
- Older versions used a cable that snaked from machine to machine, but now twisted pairs to hubs and switches are most common.
- Speeds have risen from 10 Mbps to 1 Gbps and are still rising.

Lesson 11

Wireless LANs and Bluetooth

Contents

11.0 Aim

To have an introduction about the various protocols those are being used for Wireless LANs. Moreover, the issues regarding the use of wireless networks for broad-band are also discussed. This chapter also provides an introduction about Bluetooth.

11.1 Introduction

Due to the increase in the number of systems that are being connected to networks, the importance of Wireless LANs and Bluetooth become very essential. This is mainly because the wires consumed to lay networks are bulky and they cost too much. However, these type of network connections can be within only certain area. This chapter deals with Wireless LANs, Broadband Wireless and Bluetooth.

11.2 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.

11.2.1 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 is given in Fig. 11-1.

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 sublayers. In 802.11, the MAC (Medium Access Control) sublayer determines how the channel is allocated, that is, who gets to transmit next.

Above it is the LLC (Logical Link Control) sublayer, 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.

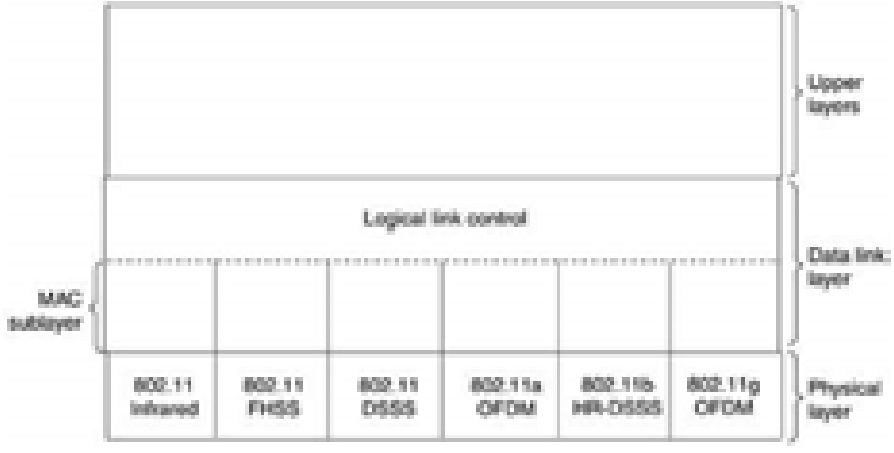


Figure 11-1. Part of the 802.11 protocol stack.

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.

11.2.2 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.

Infra Red

- ✓ 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 codeword containing fifteen 0s and a single 1, using what is called Gray code.

FHSS (Frequency Hopping Spread Spectrum)

- ✓ 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.

DSSS (Direct Sequence Spread Spectrum)

- ✓ DSSS 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 baud when 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 Frequency Division Multiplexing) to deliver up to 54 Mbps in the wider 5-GHz ISM band.
- ✓ As the term FDM suggests, different frequencies are used—52 of them, 48 for data and 4 for synchronization—not unlike ADSL.

HR-DSSS (High Rate Direct Sequence Spread Spectrum)

- ✓ It 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.

An enhanced version of 802.11b, 802.11g, was approved by IEEE in November 2001 after much politicking about whose patented technology it would use. It uses the OFDM modulation method of 802.11a but operates in the narrow 2.4-GHz ISM band along with 802.11b. In theory it can operate at up to 54 MBps. It is not yet clear whether this speed will be realized in practice. It means that the 802.11 committee has produced three different high-speed wireless LANs: 802.11a, 802.11b, and 802.11g (not to mention three low-speed wireless LANs).

11.2.3 The 802.11 MAC Sublayer Protocol

The 802.11 MAC sublayer 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 wasits 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.

To start with, there is the hidden station problem mentioned earlier and illustrated again in Fig. 11-2(a).

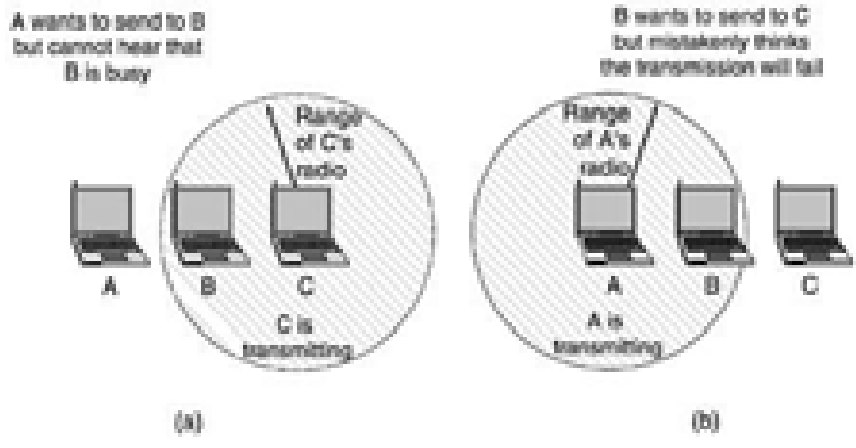


Figure 11-2. (a) The hidden station problem. (b) The exposed station problem.

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. 11-2(b). Here B wants to send to C so 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.

1. DCF (Distributed Coordination Function), does not use any kind of central control (in that respect, similar to Ethernet).
2. PCF (Point Coordination Function), uses the base station to control all activity in its cell.

11.2.4 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 sublayer. 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.

Data Frame

The format of the data frame is shown in Fig. 11-6. 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 O bit tells the receiver that a sequence of frames with this bit on must be processed strictly in order.

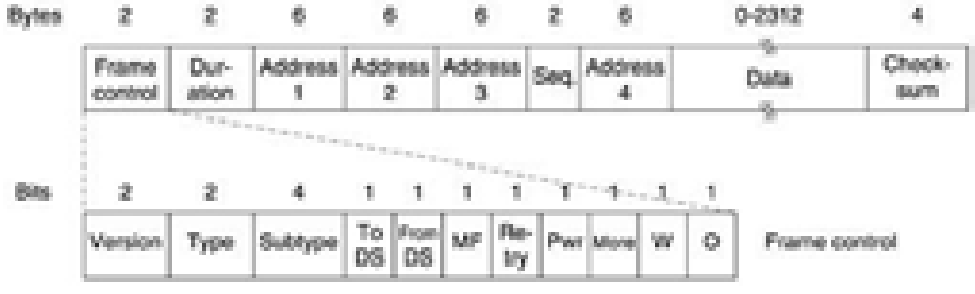


Figure 11-6. The 802.11 data frame.

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, 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

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 Frame

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.

11.2.5 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.

1. **Association.** This service is used by mobile stations to connect themselves to base stations.
2. **Disassociation.** Either the station or the base station may disassociate, thus breaking the relationship.
3. **Reassociation.** A station may change its preferred base station using this service. This facility is useful for mobile stations moving from one cell to another.
4. **Distribution.** This service determines how to route frames sent to the base station. If the destination is local to the base station, the frames can be sent out directly over the air.
5. **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 intracell (i.e., relate to actions within a single cell). They are used after association has taken place and are as follows.

1. **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.

2. **Deauthentication.** When a previously authenticated station wants to leave the network, it is deauthenticated. After deauthentication, it may no longer use the network.
3. **Privacy.** For information sent over a wireless LAN to be kept confidential, it must be encrypted. This service manages the encryption and decryption. The encryption algorithm specified is RC4, invented by Ronald Rivest of M.I.T.
4. **Data delivery.** Finally, data transmission is what it is all about, so 802.11 naturally provide a way to transmit and receive data.

An 802.11 cell has some parameters that can be inspected and, in some cases, adjusted. They relate to encryption, timeout intervals, data rates, beacon frequency, and so on.

11.3 Broadband Wireless

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 are prohibitively expensive. The best choice is broadband wireless.

Like some of the other 802 standards, 802.16 was heavily influenced by the OSI model, including the (sub) layers, terminology, service primitives, and more. Unfortunately, also like OSI, it is fairly complicated. In the following sections we will give a brief description of some of the highlights of 802.16, but this treatment is far from complete and leaves out many details.

11.3.1 Comparison of 802.11 with 802.16

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.

- 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.
- 802.16 can use full-duplex communication; something 802.11 avoids keeping 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.
- Furthermore, each cell is likely to have many more users than will a typical 802.11 cell, and these users are expected to use more bandwidth than will a typical 802.11 user. After all it is rare for a company to invite 50 employees to show up in a room with their laptops to see if they can saturate the 802.11 wireless network by watching 50 separate movies at once. For this reason, more spectrum is needed than the ISM bands can

provide, forcing 802.16 to operate in the much higher 10-to-66 GHz frequency range, the only place unused spectrum is still available.

- Another issue is quality of service. While 802.11 provides some support for real-time traffic (using PCF mode), it was not really designed for telephony and heavy-duty multimedia usage. In contrast, 802.16 is expected to support these applications completely because it is intended for residential as well as business use.

In short, 802.11 was designed to be mobile Ethernet, whereas 802.16 was designed to be wireless, but stationary, cable television. These differences are so big that the resulting standards are very different as they try to optimize different things.

11.3.2 The 802.16 Protocol Stack

The 802.16 protocol stack is illustrated in Fig. 11-7.

The general structure is similar to that of the other 802 networks, but with more sublayers. The bottom sublayer deals with transmission. Traditional narrow-band radio is used with conventional modulation schemes. Above the physical transmission layer comes a convergence sublayer 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.

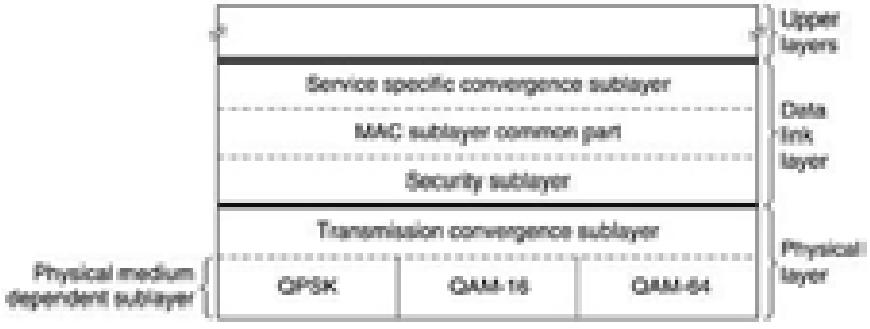


Figure 11-7. The 802.16 protocol stack.

Although not shown 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

- ✓ The data link layer consists of three sublayers.
- ✓ 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.

MAC Sub-Layer

- ✓ 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 sublayer 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 sublayer takes the place of the logical link sublayer in the other 802 protocols. Its function is to interface to the network layer.

11.3.3 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, as shown in Fig. 11-8. Each sector has its own users and is fairly independent of the adjoining ones, something not true of cellular radio, which is Omni-directional.

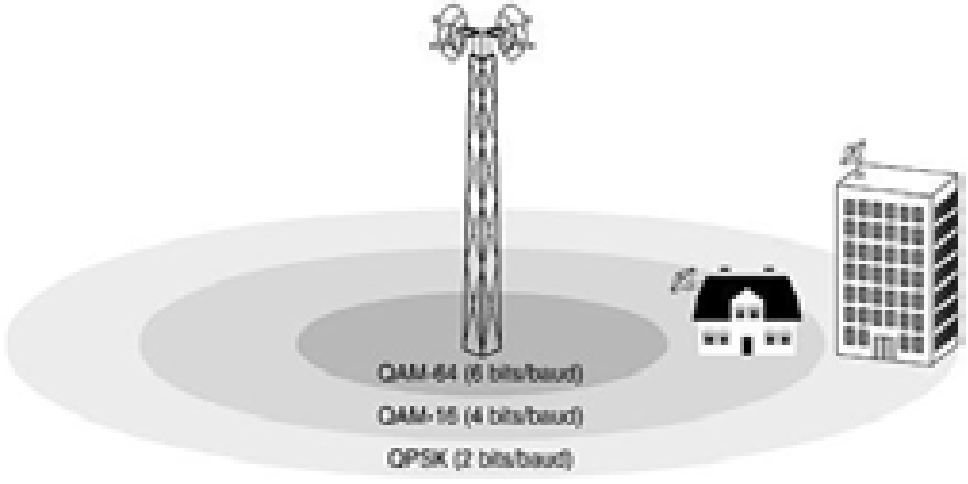


Figure 11-8. The 802.16 transmission environment.

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.

11.3.4 The 802.16 MAC Sublayer Protocol

The data link layer is divided into three sublayers, as we saw in Fig. 11-7.

Security Sub-layer

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.

MAC sublayer 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 subframe.

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:

1. Constant bit rate service.

   Constant bit rate service is intended for transmitting uncompressed voice such as on a T1 channel.

2. Real-time variable bit rate service.

   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.

3. Non-real-time variable bit rate service.

   Non-real-time variable bit rate service is for heavy transmissions that are not real time, such as large file transfers.

4. Best-efforts service.

   Finally, best-efforts service is for everything else. No polling is done and the subscriber must contend for bandwidth with other best-efforts subscribers.

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 sublayer.

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.

11.3.5 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), as illustrated in Fig. 11-10. 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?

Figure 11-10. (a) A generic frame. (b) A bandwidth request frame.

A quick rundown of the header fields of Fig. 11-10(a) is as follows.

- 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 HeaderCRC field is a checksum over the header only, using the polynomial $x^8 + x^2 + x + 1$.

A second header type, for frames that request bandwidth, is shown in Fig. 11-10(b). 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.

11.4 Bluetooth

In 1994, the L. M. Ericsson Company became interested in connecting its mobile phones to other devices (e.g., PDAs) without cables. Together with four other companies (IBM, Intel, Nokia, and Toshiba), it formed a SIG (Special Interest Group, i.e., consortium) to develop a wireless standard for interconnecting computing and communication devices and accessories using short-range, low-power, inexpensive wireless radios. The project was named Bluetooth, after Harald Blaatand (Bluetooth) II (940-981), a Viking king who unified (i.e., conquered) Denmark and Norway, also without cables.

11.4.1 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. 11-11. An interconnected collection of piconets is called a scatternet.

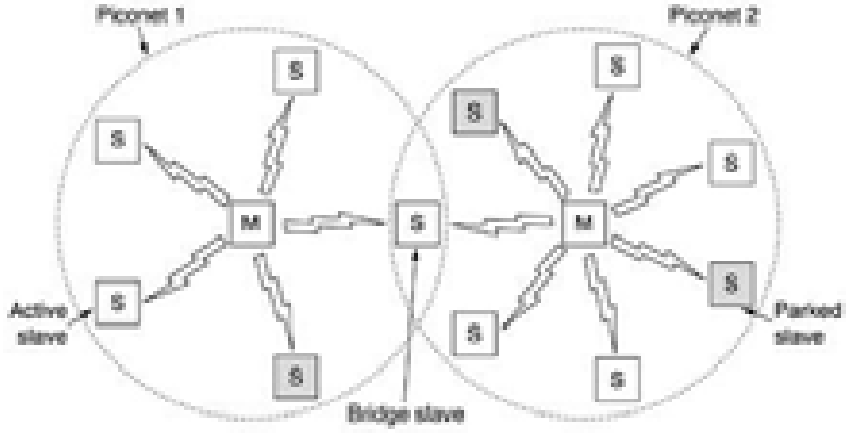


Figure 11-11. Two piconets can be connected to form a scatternet.

- 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 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.

11.4.2 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. The 13 applications, which are called profiles, are listed in Fig. 11-12.

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

Figure 11-12. The Bluetooth profiles.

11.4.3 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.

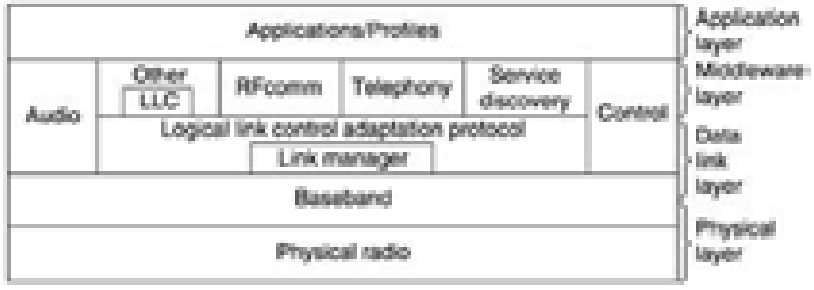


Figure 11-13. The 802.15 version of the Bluetooth protocol architecture.

The basic Bluetooth protocol architecture as modified by the 802 committee is shown in Fig. 11-13.

- 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.
- The baseband layer is somewhat analogous to the MAC sublayer 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 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 top layer is where the applications and profiles are located. They make use of the protocols in lower layers to get their work done.

In the following sections we will examine the three lowest layers of the Bluetooth protocol stack since these roughly correspond to the physical and MAC sublayers.

11.4.4 The Bluetooth Frame Structure

There are several frame formats, the most important of which is shown in Fig. 11-14.

- 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 sublayer 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.

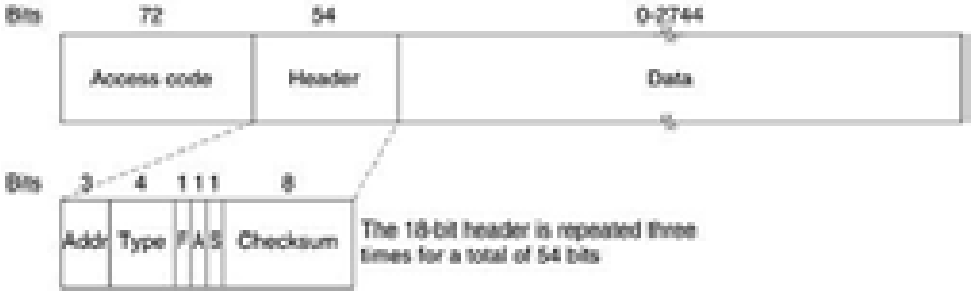


Figure 11-14. A typical Bluetooth data frame.

11.5 Summary

- Wireless LANs are becoming common, with 802.11 dominating the field.
- Its physical layer allows five different transmission modes, including infrared, various spread spectrum schemes, and a multichannel FDM system.
- The Bluetooth system is also wireless but aimed more at the desktop, for connecting headsets and other peripherals to computers without wires. It is also intended to connect peripherals, such as fax machines, to mobile telephones.
- Like 801.11, it uses frequency hopping spread spectrum in the ISM band. Due to the expected noise level of many environments and need for real-time interaction, elaborate forward error correction is built into its various protocols.

UNIT - IV

Lesson 12

Routing Algorithms – I

Contents

12.0 Aim

This lesson explains the function of the network layer. The route through which the packet is transmitted is decided in this layer. The various routing protocols for packet routing are discussed in this lesson.

12.1 Introduction

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.

12.2 Properties of a Routing Algorithm

1. Correctness
2. Simplicity
3. Robustness

4. Stability
5. Fairness and Optimality

Before we can even attempt to find trade-offs between fairness and optimality, we must decide what it is we seek to optimize. Minimizing mean packet delay is an obvious candidate, but so is maximizing total network throughput. Furthermore, these two goals are also in conflict, since operating any queueing system near capacity implies a long queueing delay.

As a compromise, many networks attempt to minimize the number of hops a packet must make, because reducing the number of hops tends to improve the delay and also reduce the amount of bandwidth consumed, which tends to improve the throughput as well.

12.3 Types of Routing Algorithms

Routing algorithms can be grouped into two major classes:

1. Non-adaptive Algorithms
2. Adaptive Algorithms

12.3.1 Non-adaptive Algorithms

Non-adaptive 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.

12.3.2 Non-adaptive Algorithms

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 □T 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.

12.4 The Optimality Principle

A general statement about optimal routes without regard to network topology or traffic 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 and is illustrated in Fig. 12.1 , where the distance metric is the number of hops.

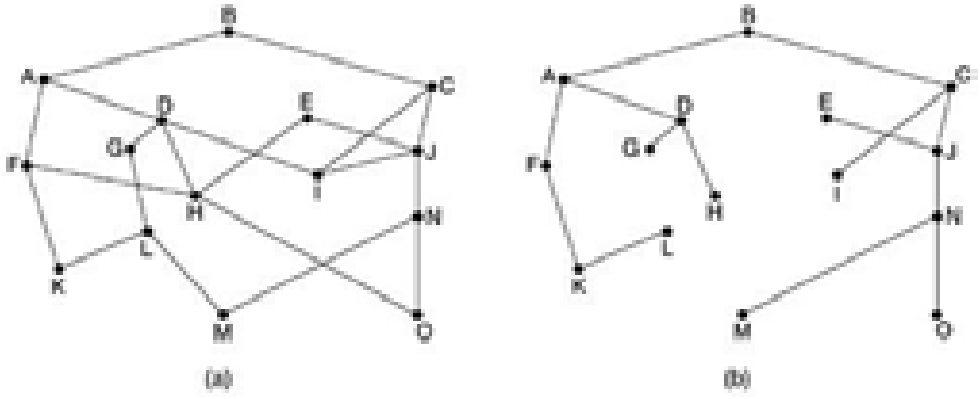


Figure 12.1 (a) A subnet. (b) A sink tree for router B.

Since a sink tree is indeed a tree, it does not contain any loops, so each packet will be delivered within a finite and bounded number of hops.

In practice, life is not quite this easy. Links and routers can go down and come back up during operation, so different routers may have different ideas about the current topology. Also, we have quietly finessed the issue of whether each router has to individually acquire the information on which to base its sink tree computation or whether this information is collected by some other means. The optimality principle and the sink tree provide a benchmark against which other routing algorithms can be measured.

12.5 Shortest Path Routing

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 in Fig. 12.2 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).

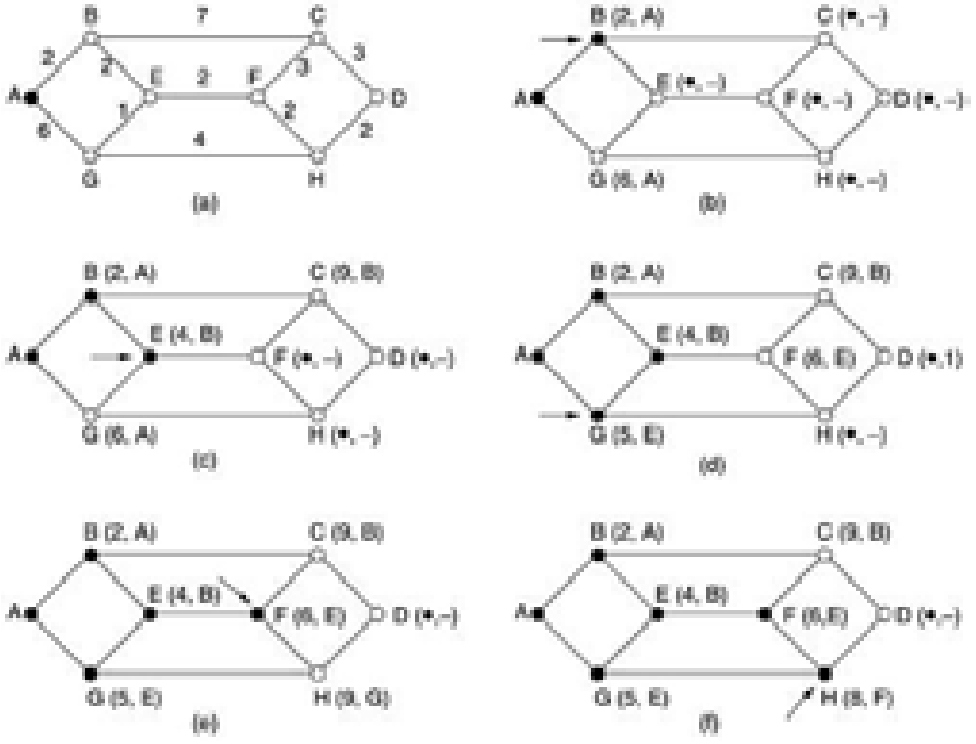


Figure 12.2. The first five steps used in computing the shortest path from A to D. The arrows indicate the working node.

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.

12.5.1 Dijkstra's Algorithm

In this algorithm, 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.

To illustrate how the labeling algorithm works, look at the weighted, undirected graph of Fig. 12.2 (a), where the weights represent, for example, distance.

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.

```
#define MAX_NODES 1024                  /* maximum number of nodes */
#define INFINITY 1000000000             /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES];/* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                        /* the path being worked on */
      int predecessor;                  /* previous node */
      int length;                       /* length from source to this node */
      enum {permanent, tentative} label; /* label state */
  } state[MAX_NODES];

  int i, k, min;
  struct state *p;

  for (p = &state[0]; p < &state[n]; p++) { /* initialize state */
      p->predecessor = -1;
      p->length = INFINITY;
      p->label = tentative;
  }
  state[t].length = 0;  state[t].label = permanent;
  k = t;                                /* k is the initial working node */
  do {                                  /* Is there a better path from k? */
      for (i = 0; i < n; i++)           /* this graph has n nodes */
          if (dist[k][i] != 0 && state[i].label == tentative) {
              if (state[k].length + dist[k][i] < state[i].length) {
                  state[i].predecessor = k;
                  state[i].length = state[k].length + dist[k][i];
              }
          }

      /* Find the tentatively labeled node with the smallest label. */
      k = 0; min = INFINITY;
      for (i = 0; i < n; i++)
          if (state[i].label == tentative && state[i].length < min) {
              min = state[i].length;
              k = i;
          }
      state[k].label = permanent;
  } while (k != s);

  /* Copy the path into the output array. */
  i = 0;  k = s;
  do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

Figure 12.3. Dijkstra's algorithm to compute the shortest path through a graph.

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, as shown in Fig. 12.2 (b). 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. Fig. 12.2 shows the first five steps of the algorithm.

To see why the algorithm works, look at Fig. 12.2 (c). At that point we have just made E permanent. 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. This algorithm is given in Fig. 12.3. The global variables n and dist describe the graph and are initialized before shortest path is called.

The only difference between the program and the algorithm described above is that in Fig. 12.3, we compute the shortest path starting at the terminal node, t, rather than at the source node, s. Since the shortest path from t to s in an undirected graph is the same as the shortest path from s to t, it does not matter at which end we begin (unless there are several shortest paths, in which case reversing the search might discover a different one).

The reason for searching backward is that each node is labeled with its predecessor rather than its successor. When the final path is copied into the output variable, path, the path is thus reversed. By reversing the search, the two effects cancel, and the answer is produced in the correct order.

12.6 Flooding

It is a static algorithm, in which every incoming packet is sent out on every outgoing line except the one it arrived on. Flooding obviously generates vast numbers of duplicate packets, in fact, an infinite number unless some measures are taken to damp the process. One such measure is to have a hop counter contained in the header of each packet, which is decremented at each hop, with the packet being discarded when the counter reaches zero. Ideally, the hop counter should be initialized to the length of the path from source to destination. If the sender does not know how long the path is, it can initialize the counter to the worst case, namely, the full diameter of the subnet.

An alternative technique for damming the flood is to keep track of which packets have been flooded, to avoid sending them out a second time. To achieve this goal is to have the source router put a sequence number in each packet it receives from its hosts. Each router then needs a list per source router telling which sequence numbers originating at that source have already been seen. If an incoming packet is on the list, it is not flooded. To prevent the list from

growing without bound, each list should be augmented by a counter, k, meaning that all sequence numbers through k have been seen. When a packet comes in, it is easy to check if the packet is a duplicate; if so, it is discarded. Furthermore, the full list below k is not needed, since k effectively summarizes it.

12.6.1 Selective Flooding

A variation of flooding that is slightly more practical is selective flooding. In this algorithm the routers do not send every incoming packet out on every line, only on those lines that are going approximately in the right direction. There is usually little point in sending a westbound packet on an eastbound line unless the topology is extremely peculiar and the router is sure of this fact.

12.7 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.

Distance vector routing algorithms operate by having each router maintain a table (i.e., a vector) 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. His 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.

Example

Assume that delay is used as a metric and that the router knows the delay to each of its neighbor. Once every T msec each router sends to each neighbor a list of its estimated delays to each destination. It also receives a similar list from each neighbor. Imagine that one of these tables has just come in from neighbor X, with X_i being X's estimate of how long it takes to get to router i.

If the router knows that the delay to X is m msec, it also knows that it can reach router i via X in X_i + m msec. By performing this calculation for each neighbor, a router can find out which estimate seems the best and use that estimate and the corresponding line in its new routing table.

Note that the old routing table is not used in the calculation. This updating process is illustrated in Fig. 12.4 Part (a) shows a subnet. The first four columns of part (b) show the delay vectors received from the neighbors of router J. A claims to have a 12-msec delay to B, a 25-msec delay to C, a 40-msec delay to D, etc. Suppose that J has measured or estimated its delay to its neighbors, A, I, H, and K as 8, 10, 12, and 6 msec, respectively.

Consider how J computes its new route to router G. It knows that it can get to A in 8 msec, and A claims to be able to get to G in 18 msec, so J knows it can count on a delay of 26 msec to G if it forwards packets bound for G to A. Similarly, it computes the delay to G via I, H, and K as 41 (31 + 10), 18 (6 + 12), and 37 (31 + 6) msec, respectively.

The best of these values is 18, so it makes an entry in its routing table that the delay to G is 18 msec and that the route to use is via H. The same calculation is performed for all the other destinations, with the new routing table shown in the last column of the figure.

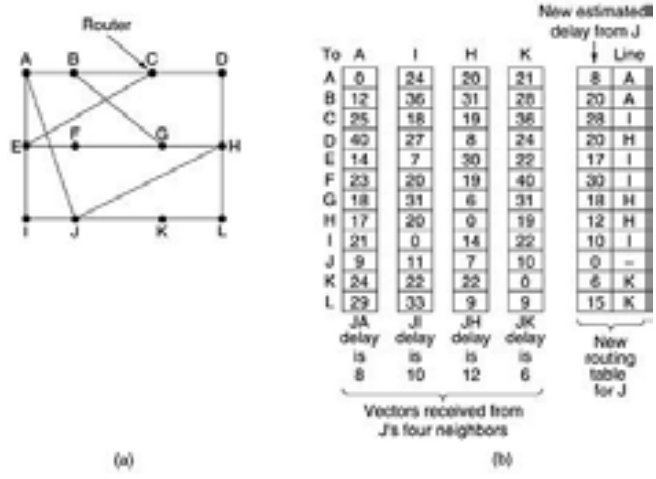


Fig. 12.4 (a) A subnet. (b) Input from A, I, H, K, and the new routing table for J.

12.8 Link State Routing

Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing because of two primary problems.

1. 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.
2. The algorithm often took too long to converge (the count-to-infinity problem).

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

1. Discover its neighbors and learn their network addresses.
2. Measure the delay or cost to each of its neighbors.

3. Construct a packet telling all it has just learned.
4. Send this packet to all other routers.
5. 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.

12.8.1 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.When two or more routers are connected by a LAN, the situation is slightly more complicated. Fig. 12.5 (a) illustrates a LAN to which three routers, A, C, and F, are directly connected. Each of these routers is connected to one or more additional routers, as shown.

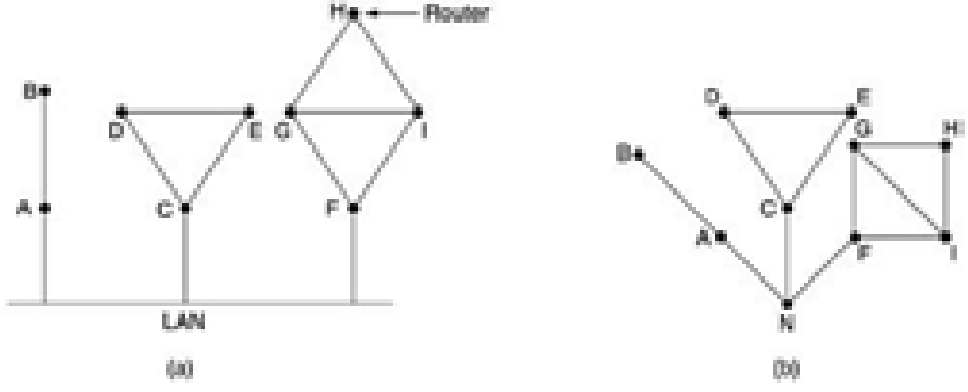


Fig. 12.5 (a) Nine routers and a LAN. (b) A graph model of (a).

One way to model the LAN is to consider it as a node itself, as shown in Fig. 12.5 (b). 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.

12.8.2 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.

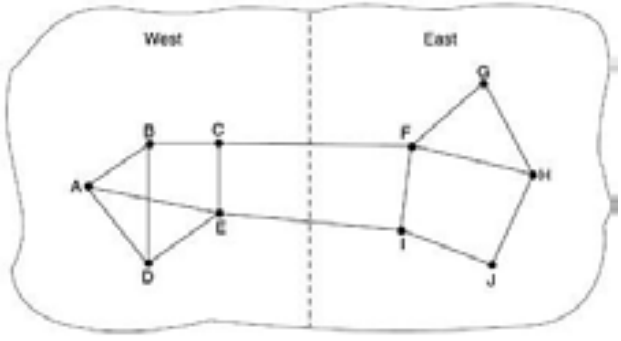


Figure 12.6 A subnet in which the East and West parts are connected by two lines

Suppose that most of the traffic between East and West is using line CF, and as a result, this line is heavily loaded with long delays. Including queuing delay in the shortest path calculation will make EI more attractive. After the new routing tables have been installed, most of the East-West traffic will now go over EI, overloading this line.

Consequently, in the next update, CF will appear to be the shortest path. As a result, the routing tables may oscillate wildly, leading to erratic routing and many potential problems. If load is ignored and only bandwidth is considered, this problem does not occur. Alternatively, the load can be spread over both lines, but this solution does not fully utilize the best path.

Nevertheless, to avoid oscillations in the choice of best path, it may be wise to distribute the load over multiple lines, with some known fraction going over each line.

12.8.3 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. An example subnet is given in Fig. 12.7 (a) with delays shown as labels on the lines. The corresponding link state packets for all six routers are shown in **Fig. 12.7 (b)**.

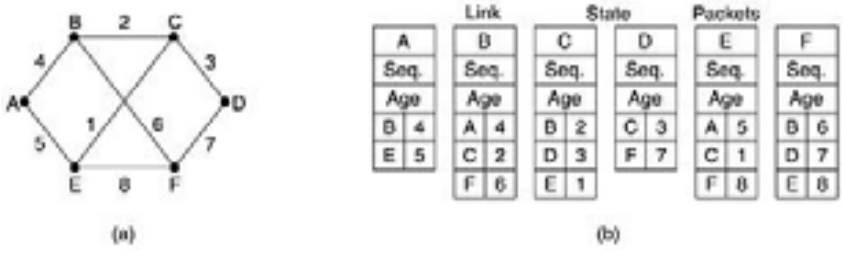


Fig. 12.7 (a) A subnet. (b) The link state packets for this subnet.

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.

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.

The solution to all these problems is to include the age of each packet after the sequence number and decrement it once per second. When the age hits zero, the information from that router is discarded.

Normally, a new packet comes in, say, every 10 sec, so router information only times out when a router is down (or six consecutive packets have been lost, an unlikely event).

The Age field is also decremented by each router during the initial flooding process, to make sure no packet can get lost and live for an indefinite period of time (a packet whose age is zero is discarded).

Some refinements to this algorithm make it more robust.

When a link state packet comes in to a router for flooding, it is not queued for transmission immediately. Instead it is first put in a holding area to wait a short while. If another link state packet from the same source comes in before the first packet is transmitted, their sequence numbers are compared.

If they are equal, the duplicate is discarded. If they are different, the older one is thrown out. To guard against errors on the router-router lines, all

link state packets are acknowledged. When a line goes idle, the holding area is scanned in round-robin order to select a packet or acknowledgement to send.

The data structure used by router B for the subnet shown in Fig. 12.7 (a) is depicted in Fig. 12.7 Each row here corresponds to a recently-arrived, but as yet not fully-processed, link state packet. The table records where the packet originated, its sequence number and age, and the data. In addition, there are send and acknowledgement flags for each of B's three lines (to A, C, and F, respectively).

The send flags mean that the packet must be sent on the indicated line. The acknowledgement flags mean that it must be acknowledged there.

			Send flags			ACK flags			
Source	Seq.	Age	A	C	F	A	C	F	Data
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

Figure 12-8. The packet buffer for router B in Fig 12-7.

In Fig 12-8, the link state packet from A arrives directly, so it must be sent to C and F and acknowledged to A, as indicated by the flag bits. Similarly, the packet from F has to be forwarded to A and C and acknowledged to F.

However, the situation with the third packet, from E, is different. It arrived twice, once via EAB and once via EFB. Consequently, it has to be sent only to C but acknowledged to both A and F, as indicated by the bits.

If a duplicate arrives while the original is still in the buffer, bits have to be changed. For example, if a copy of C's state arrives from F before the fourth entry in the table has been forwarded, the six bits will be changed to 100011 to indicate that the packet must be acknowledged to F but not sent there.

12.8.4 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. However, problems with the hardware or software can wreak havoc with this algorithm (also with other ones). For example, if a router claims to have a line it does not have or forgets a line it does have, the subnet graph will be incorrect.

If a router fails to forward packets or corrupts them while forwarding them, trouble will arise. Finally, if it runs out of memory or does the routing calculation wrong, bad things will happen. As the subnet grows into the range of tens or hundreds of thousands of nodes, the probability of some router failing occasionally becomes non-negligible.

The trick is to try to arrange to limit the damage when the inevitable happens. Link state routing is widely used in actual networks, so a few words about some example protocols using it are in order.

The OSPF protocol, which is widely used in the Internet, uses a link state algorithm. 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. Basically IS-IS distributes a picture of the router topology, from which the shortest paths are computed. Each router announces, in its link state information, which network layer addresses it can reach directly.

These addresses can be IP, IPX, AppleTalk, or any other addresses. IS-IS can even support multiple network layer protocols at the same time. Many of the innovations designed for IS-IS were adopted by OSPF (OSPF was designed several years after IS-IS). These include a self-stabilizing method of flooding link state updates, the concept of a designated router on a LAN, and the method of computing and supporting path splitting and multiple metrics.

As a consequence, there is very little difference between IS-IS and OSPF. The most important difference is that IS-IS is encoded in such a way that it is easy and natural to simultaneously carry information about multiple network layer protocols, a feature OSPF does not have. This advantage is especially valuable in large multiprotocol environments.

12.9 Summary

1. Router as two processes one of them handles each packet as it arrives, looking up the outgoing line to use for it in the routing tables. This process is forwarding. The other process is responsible for filling in and updating the routing tables.

2. Adaptive and non-adaptive differs in adapting to the network traffic.

Lesson 13

ROUTING ALGORITHMS – II

Contents

13.1 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. 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. For huge networks, a two-level hierarchy may be insufficient; it may be necessary to group the regions into clusters, the clusters into zones, the zones into groups, and so on, until we run out of names for aggregations.

Example of a multilevel hierarchy

Fig. 13-1 gives a quantitative example of routing in a two-level hierarchy with five regions. The full routing table for router 1A has 17 entries, as shown in Fig. 13-1(b).When routing is done hierarchically, as in Fig. 13-1(c), there are entries for all the local routers as before, but all other regions have been condensed into a single router, so all traffic for region 2 goes via the 1B -2A line, but the rest of the remote traffic goes via the 1C -3B line.

Hierarchical routing has reduced the table from 17 to 7 entries. As the ratio of the number of regions to the number of routers per region grows, the savings in table space increase.

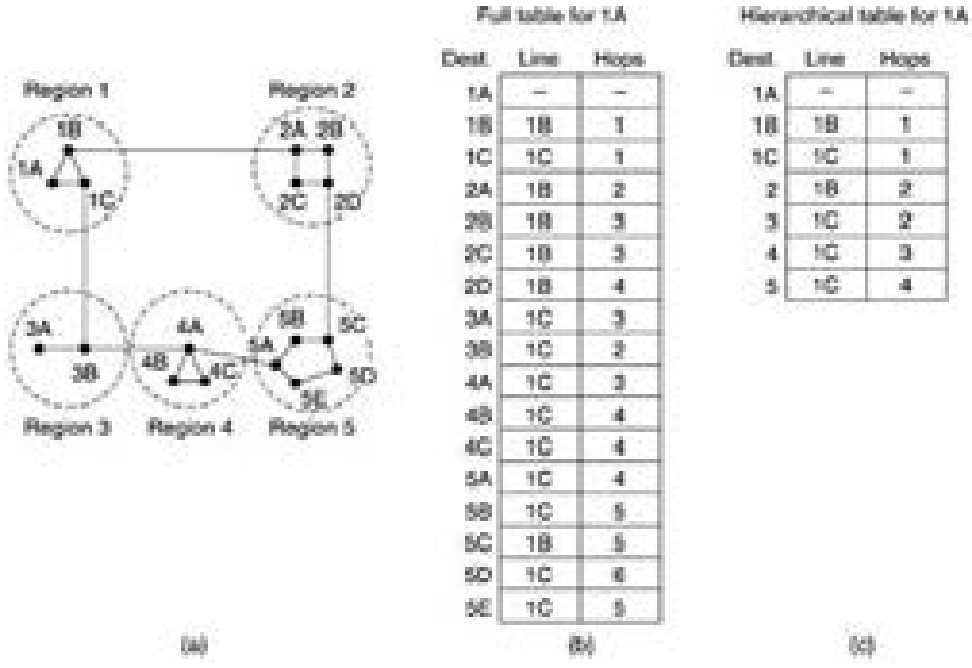


Figure 13-1. Hierarchical routing.

Unfortunately, these gains in space are not free. There is a penalty to be paid, and this penalty is in the form of increased path length. For example, the best route from 1A to 5C is via region 2, but with hierarchical routing all traffic to region 5 goes via region 3, because that is better for most destinations in region 5.

13.2 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.

Method 1

- ✓ 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.

Method 2

- ✓ 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.

Method 3 (Multi-destination routing)

- ✓ 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.

Method 4

- ✓ 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).

Method 5

- ✓ 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.
- ✓ An example of reverse path forwarding is shown in Fig. 13-2. Part (a) shows a subnet, part (b) shows a sink tree for router I of that subnet, and part (c) shows how the reverse path algorithm works.
- ✓ On the first hop, I sends packets to F, H, J, and N, as indicated by the second row of the tree. Each of these packets arrives on the preferred path to I (assuming that the preferred path falls along the sink tree) and is so indicated by a circle around the letter.
- ✓ On the second hop, eight packets are generated, two by each of the routers that received a packet on the first hop.
- ✓ As it turns out, all eight of these arrive at previously unvisited routers, and five of these arrive along the preferred line.
- ✓ Of the six packets generated on the third hop, only three arrive on the preferred path (at C, E, and K); the others are duplicates.
- ✓ After five hops and 24 packets, the broadcasting terminates, compared with four hops and 14 packets had the sink tree been followed exactly.

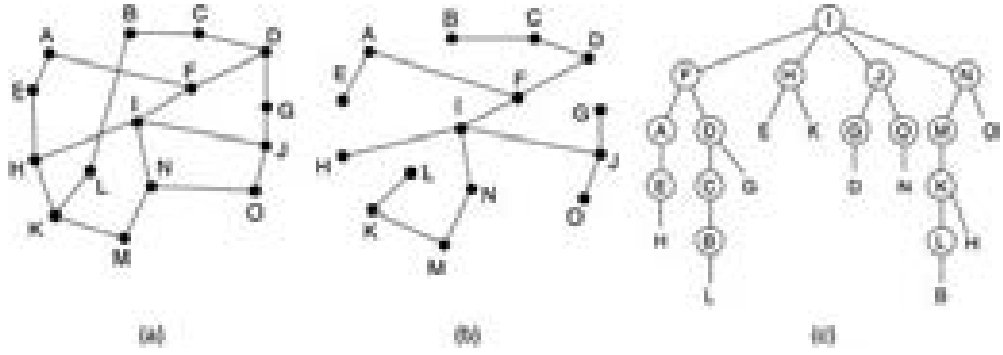


Figure 13-2. Reverse path forwarding. (a) A subnet. (b) A sink tree. (c) The tree built by reverse path forwarding.

- ✓ The principal advantage of reverse path forwarding is that it is both reasonably efficient and easy to implement.
- ✓ It does not require routers to know about spanning trees, nor does it have the overhead of a destination list or bit map in each broadcast packet as does multi-destination addressing.
- ✓ Nor does it require any special mechanism to stop the process, as flooding does (either a hop counter in each packet and a priori knowledge of the subnet diameter, or a list of packets already seen per source).

13.3 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. 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. For example, in Fig. 13-3(a) 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 is shown in Fig. 13-3(b).

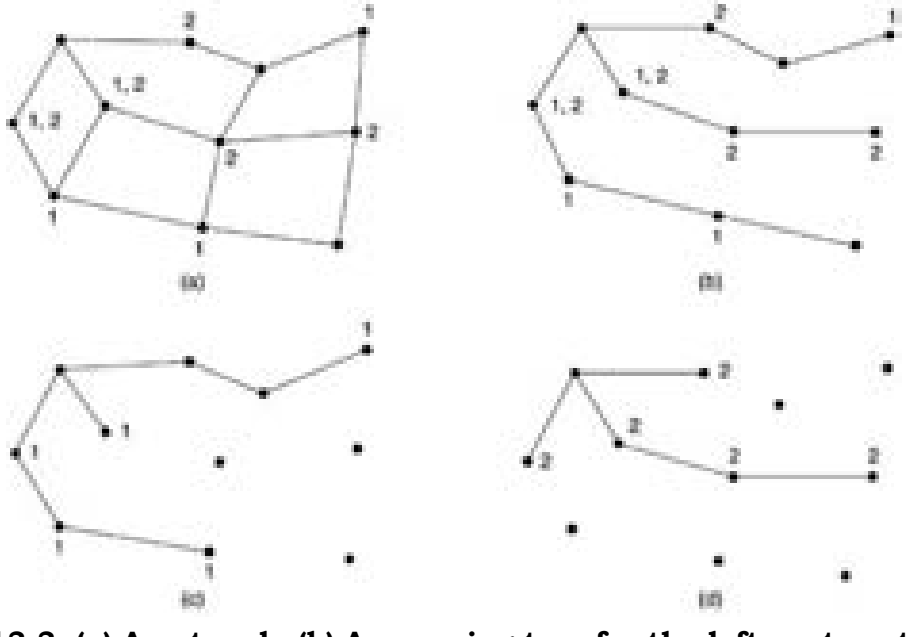


Figure 13-3. (a) A network. (b) A spanning tree for the leftmost router. (c) A multicast tree for group 1. (d) A multicast tree for group 2.

13.4 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 mobile hosts 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.The model of the world that network designers typically use is shown in Fig. 13-4. Here we have a WAN consisting of routers and hosts.

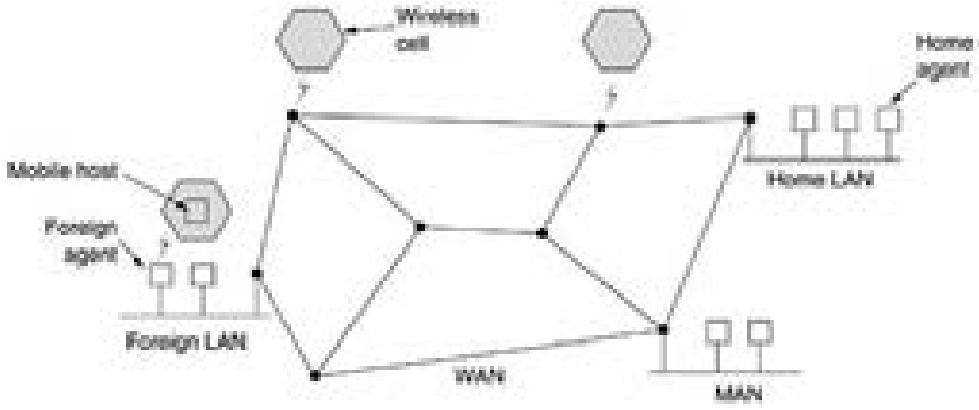


Figure 13-4. A WAN to which LANs, MANs, and wireless cells are attached.

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. All hosts are assumed to have a permanent home location that never changes.

The routing goal in systems with mobile hosts is to make it possible to send packets to mobile hosts using their home addresses and have the packets efficiently reach them wherever they may be. The trick, of course, is to find them. In the model of Fig. 13-4, the world is divided up (geographically) into small units. Let us call them areas, where an area is typically a LAN or wireless cell. Each area has one or more foreign agents, which are processes that keep track of all mobile hosts visiting the area.

In addition, each area has a home agent, which keeps track of hosts whose home is in the area, but who are currently visiting another area.

When a new host enters an area, either by connecting to it (e.g., plugging into the LAN) or just wandering into the cell, his computer must register itself with the foreign agent there. The registration procedure typically works like this:

1. Periodically, each foreign agent broadcasts a packet announcing its existence and address. A newly-arrived mobile host may wait for one of these messages, but if none arrives quickly enough, the mobile host can broadcast a packet saying: Are there any foreign agents around?
2. The mobile host registers with the foreign agent, giving its home address, current data link layer address, and some security information.
3. The foreign agent contacts the mobile host's home agent and says: One of your hosts is over here. The message from the foreign agent to the home agent contains the foreign agent's network address. It also includes the security information to convince the home agent that the mobile host is really there.
4. The home agent examines the security information, which contains a timestamp, to prove that it was generated within the past few seconds. If it is happy, it tells the foreign agent to proceed.
5. When the foreign agent gets the acknowledgement from the home agent, it makes an entry in its tables and informs the mobile host that it is now registered.

Ideally, when a host leaves an area, that, too, should be announced to allow deregistration, but many users abruptly turn off their computers when done. When a packet is sent to a mobile host, it is routed to the host's home LAN because that is what the address says should be done, as illustrated in step 1 of Fig. 13-5.

Here the sender, in the northwest city of Seattle, wants to send a packet to a host normally across the United States in New York. Packets sent to the mobile host on its home LAN in New York are intercepted by the home agent there.

The home agent then looks up the mobile host's new (temporary) location and finds the address of the foreign agent handling the mobile host, in Los Angeles.

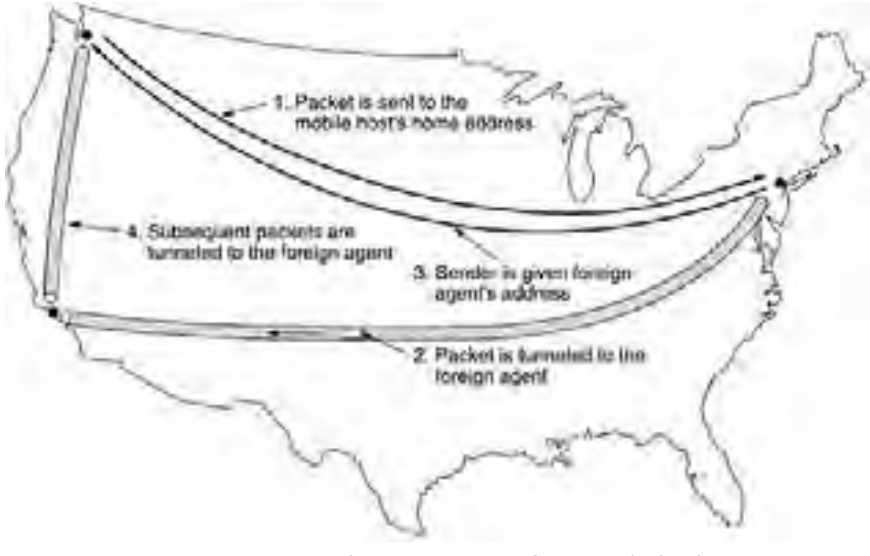


Figure 13-5. Packet routing for mobile hosts.

- The home agent then does two things.
    1. First, it encapsulates the packet in the payload field of an outer packet and sends the latter to the foreign agent (step 2 in Fig. 13-5). This mechanism is called tunneling; we will look at it in more detail later. After getting the encapsulated packet, the foreign agent removes the original packet from the payload field and sends it to the mobile host as a data link frame.
    2. Second, the home agent tells the sender to henceforth send packets to the mobile host by encapsulating them in the payload of packets explicitly addressed to the foreign agent instead of just sending them to the mobile host's home address (step 3). Subsequent packets can now be routed directly to the host via the foreign agent (step 4), bypassing the home location entirely.
- The various schemes that have been proposed differ in several ways.
    1. First, there is the issue of how much of this protocol is carried out by the routers and how much by the hosts, and in the latter case, by which layer in the hosts.
    2. Second, in a few schemes, routers along the way record mapped addresses so they can intercept and redirect traffic even before it gets to the home location.
    3. Third, in some schemes each visitor is given a unique temporary address; in others, the temporary address refers to an agent that handles traffic for all visitors.
    4. Fourth, the schemes differ in how they actually manage to arrange for packets that are addressed to one destination to be delivered to a different one. One choice is changing the destination address and just retransmitting the modified packet. Alternatively, the whole packet, home address and all, can be encapsulated inside the payload of another packet sent to the temporary address.
    5. Finally, the schemes differ in their security aspects. In general, when a host or router gets a message of the form "Starting right now, please send all of Stephany's mail to me," it might have a couple of questions about whom it was talking to and whether this is a good idea

13.5 Routing in Ad Hoc Networks

There are cases in which the routers are mobile. Among the possibilities are:

1. Military vehicles on a battlefield with no existing infrastructure.
2. A fleet of ships at sea.
3. Emergency workers at an earthquake that destroyed the infrastructure.
4. 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 hoc networks or MANETs (Mobile Ad hoc NETworks).

The following makes ad-hoc networks different from wired networks:

- Routers can come and go or appear in new places at the drop of a bit. With a wired network, if a router has a valid path to some destination, that path continues to be valid indefinitely (barring a failure somewhere in the system).
- With an ad hoc network, the topology may be changing all the time, so desirability and even validity of paths can change spontaneously, without warning.

Needless to say, these circumstances make routing in ad hoc networks quite different from routing in their fixed counterparts.

13.6 AODV (Ad hoc On-demand Distance Vector) Routing Algorithm

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. Let us now see what that means.

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.

To describe the algorithm, consider the ad hoc network of Fig. 13-6, in which a process at node A wants to send a packet to node I. The AODV algorithm maintains a table at each node, keyed by destination, giving information about that destination, including which neighbor to send packets to in order to reach the destination.

Suppose that A looks in its table and does not find an entry for I. It now has to discover a route to I. This property of discovering routes only when they are needed is what makes this algorithm ''on demand.''

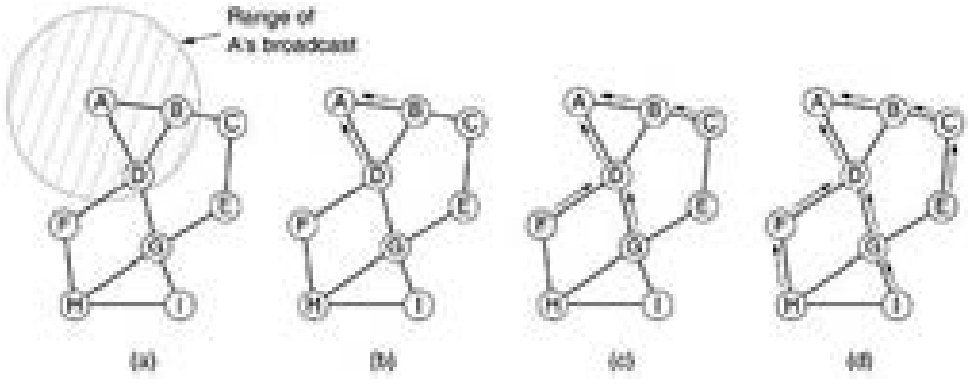


Figure 13-6. (a) Range of A's broadcast. (b) After B and D have received A's broadcast. (c) After C, F, and G have received A's broadcast. (d) After E, H, and I have received A's broadcast. The shaded nodes are new recipients. The arrows show the possible reverse routes.

To locate I, A constructs a special ROUTE REQUEST packet and broadcasts it. The packet reaches B and D, as illustrated in Fig. 13-6(a). In fact, the reason B and D are connected to A in the graph is that they can receive communication from A. F, for example, is not shown with an arc to A because it cannot receive A's radio signal. Thus, F is not connected to A.

The format of the ROUTE REQUEST packet is shown in Fig. 13-7. It contains the source and destination addresses, typically their IP addresses, which identify who is looking for whom. It also contains a Request ID, which is a local counter maintained separately by each node and incremented each time a ROUTE REQUEST is broadcast.

Together, the Source address and Request ID fields uniquely identify the ROUTE REQUEST packet to allow nodes to discard any duplicates they may receive.

Figure 13-7. Format of a ROUTE REQUEST packet.

In addition to the Request ID counter, each node also maintains a second sequence counter incremented whenever a ROUTE REQUEST is sent (or a reply to someone else's ROUTE REQUEST). It functions a little bit like a clock and is used to tell new routes from old routes.

The fourth field of Fig. 13-7 is A's sequence counter; the fifth field is the most recent value of I's sequence number that A has seen (0 if it has never seen it). The use of these fields will become clear shortly. The final field, Hop count, will keep track of how many hops the packet has made. It is initialized to 0.

When a ROUTE REQUEST packet arrives at a node (B and D in this case), it is processed in the following steps.

1. The (Source address, Request ID) pair is looked up in a local history table to see if this request has already been seen and processed. If it is a duplicate, it is discarded and processing stops. If it is not a duplicate, the pair is entered into the history table so future duplicates can be rejected, and processing continues.

2. The receiver looks up the destination in its route table. If a fresh route to the destination is known, a ROUTE REPLY packet is sent back to the source telling it how to get to the destination (basically: Use me). Fresh means that the Destination sequence number stored in the routing table is greater than or equal to the Destination sequence number in the ROUTE REQUEST packet. If it is less, the stored route is older than the previous route the source had for the destination, so step 3 is executed.
3. Since the receiver does not know a fresh route to the destination, it increments the Hop count field and rebroadcasts the ROUTE REQUEST packet. It also extracts the data from the packet and stores it as a new entry in its reverse route table. This information will be used to construct the reverse route so that the reply can get back to the source later. The arrows in Fig. 13-6 are used for building the reverse route. A timer is also started for the newly-made reverse route entry. If it expires, the entry is deleted.

Neither B nor D knows where I is, so each of them creates a reverse route entry pointing back to A, as shown by the arrows in Fig. 13-6, and broadcasts the packet with Hop count set to 1. The broadcast from B reaches C and D. C makes an entry for it in its reverse route table and rebroadcasts it. In contrast, D rejects it as a duplicate.

Similarly, D's broadcast is rejected by B. However, D's broadcast is accepted by F and G and stored, as shown in Fig. 13-6(c). After E, H, and I receive the broadcast, the ROUTE REQUEST finally reaches a destination that knows where I is, namely, I itself, as illustrated in Fig. 13-6(d).

Note that although we have shown the broadcasts in three discrete steps here, the broadcasts from different nodes are not coordinated in any way. In response to the incoming request, I builds a ROUTE REPLY packet, as shown in Fig. 13-8. The Source address, Destination address, and Hop count are copied from the incoming request, but the Destination sequence number taken from its counter in memory.

The Hop count field is set to 0. The Lifetime field controls how long the route is valid. This packet is unicast to the node that the ROUTE REQUEST packet came from, in this case, G.

It then follows the reverse path to D and finally to A. At each node, Hop count is incremented so the node can see how far from the destination (I) it is.

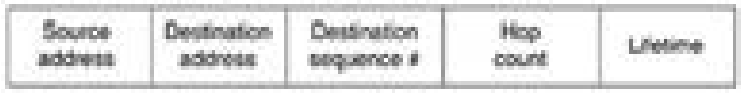


Figure 13-8. Format of a ROUTE REPLY packet.

At each intermediate node on the way back, the packet is inspected. It is entered into the local routing table as a route to I if one or more of the following three conditions are met:

1. No route to I is known.
2. The sequence number for I in the ROUTE REPLY packet is greater than the value in the routing table.

3. The sequence numbers are equal but the new route is shorter.

In this way, all the nodes on the reverse route learn the route to I for free, as a byproduct of A's route discovery. Nodes that got the original REQUEST ROUTE packet but were not on the reverse path (B, C, E, F, and H in this example) discard the reverse route table entry when the associated timer expires. In a large network, the algorithm generates many broadcasts, even for destinations that are close by.

The number of broadcasts can be reduced as follows. The IP packet's Time to live is initialized by the sender to the expected diameter of the network and decremented on each hop. If it hits 0, the packet is discarded instead of being broadcast. The discovery process is then modified as follows. To locate a destination, the sender broadcasts a ROUTE REQUEST packet with Time to live set to 1.

If no response comes back within a reasonable time, another one is sent, this time with Time to live set to 2. Subsequent attempts use 3, 4, 5, etc. In this way, the search is first attempted locally, then in increasingly wider rings.

Route Maintenance

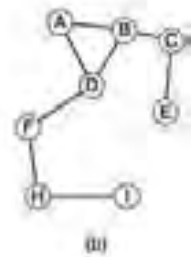
Because nodes can move or be switched off, the topology can change spontaneously. For example, in Fig. 13-6, if G is switched off, A will not realize that the route it was using to I (ADGI) is no longer valid. The algorithm needs to be able to deal with this. Periodically, each node broadcasts a Hello 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. A possible routing table for node D in our example topology is shown in Fig. 13-9.

Dest	Next hop	Distance	Active neighbors	Other fields
A	A	1	F, G	
B	B	1	F, G	
C	B	2	F	
F	G	2		
F	F	1	A, B	
G	G	1	A, B	
H	F	2	A, B	
I	G	2	A, B	

(a)



(b)

Figure 13-9. (a) D's routing table before G goes down. (b) The graph after G has gone down.

When any of N's neighbors becomes unreachable, it checks its routing table to see which destinations have routes using the now-gone neighbor. For each of these routes, the active neighbors are informed that their route via N is now invalid and must be purged from their routing tables.

The active neighbors then tell their active neighbors, and so on, recursively, until all routes depending on the now-gone node are purged from all routing tables. As an example of route maintenance, consider our previous example, but now with G suddenly switched off. The changed topology is illustrated in Fig. 13-9(b). When D discovers that G is gone, it looks at its routing table and sees that G was used on routes to E, G, and I.

The union of the active neighbors for these destinations is the set {A, B}. In other words, A and B depend on G for some of their routes, so they have to be informed that these routes no longer work. D tells them by sending them packets that cause them to update their own routing tables accordingly. D also purges the entries for E, G, and I from its routing table.

It may not have been obvious from our description, but a critical difference between AODV and Bellman-Ford is that nodes do not send out periodic broadcasts containing their entire routing table. This difference saves both bandwidth and battery life. AODV is also capable of doing broadcast and multicast routing.

Node Lookup in Peer-to-Peer Networks

A relatively new phenomenon is peer-to-peer networks, in which a large number of people, usually with permanent wired connections to the Internet, are in contact to share resources. The first widespread application of peer-to-peer technology was for mass crime: 50 million Napster users were exchanging copyrighted songs without the copyright owners' permission until Napster was shut down by the courts amid great controversy.

Nevertheless, peer-to-peer technology has many interesting and legal uses. It also has something similar to a routing problem, although it is not quite the same as the ones we have studied so far. Nevertheless, it is worth a quick look. What makes peer-to-peer systems interesting is that they are totally distributed. All nodes are symmetric and there is no central control or hierarchy.

In a typical peer-to-peer system the users each have some information that may be of interest to other users. This information may be free software, (public domain) music, photographs, and so on. If there are large numbers of users, they will not know each other and will not know where to find what they are looking for.

One solution is a big central database, but this may not be feasible for some reason (e.g., nobody is willing to host and maintain it). Thus, the problem comes down to how a user finds a node that contains what he is looking for in the absence of a centralized database or even a centralized index. Let us assume that each user has one or more data items such as songs, photographs, programs, files, and so on that other users might want to read. Each item has an ASCII string naming it. A potential user knows just the ASCII string and wants to find out if one or more people have copies and, if so, what their IP addresses are.

As an example, consider a distributed genealogical database. Each genealogist has some on-line records for his or her ancestors and relatives, possibly with photos, audio, or even video clips of the person. Multiple people may have the same great grandfather, so an ancestor may have records at multiple nodes. The name of the record is the person's name in some canonical form. At some point, a genealogist discovers his great grandfather's will in an archive, in which the great grandfather bequeaths his gold pocket watch to his nephew.

The genealogist now knows the nephew's name and wants to find out if any other genealogist has a record for him. How, without a central database, do we find out who, if anyone, has records? Various algorithms have been proposed to solve this problem, a famous one of which is Chord.

Chord System

The Chord system consists of n participating users, each of whom may have some stored records and each of whom is prepared to store bits and pieces of the index for use by other users.

Each user node has an IP address that can be hashed to an m-bit number using a hash function, hash. Chord uses SHA-1 for hash, which is just a function that takes a variable-length byte string as argument and produces a highly-random 160-bit number. Thus, we can convert any IP address to a 160-bit number called the node identifier.

Conceptually, all the 2^{160} node identifiers are arranged in ascending order in a big circle. Some of them correspond to participating nodes, but most of them do not. In Fig. 13-10(a) we show the node identifier circle for m = 5 (just ignore the arcs in the middle for the moment). In this example, the nodes with identifiers 1, 4, 7, 12, 15, 20, and 27 correspond to actual nodes and are shaded in the figure; the rest do not exist.

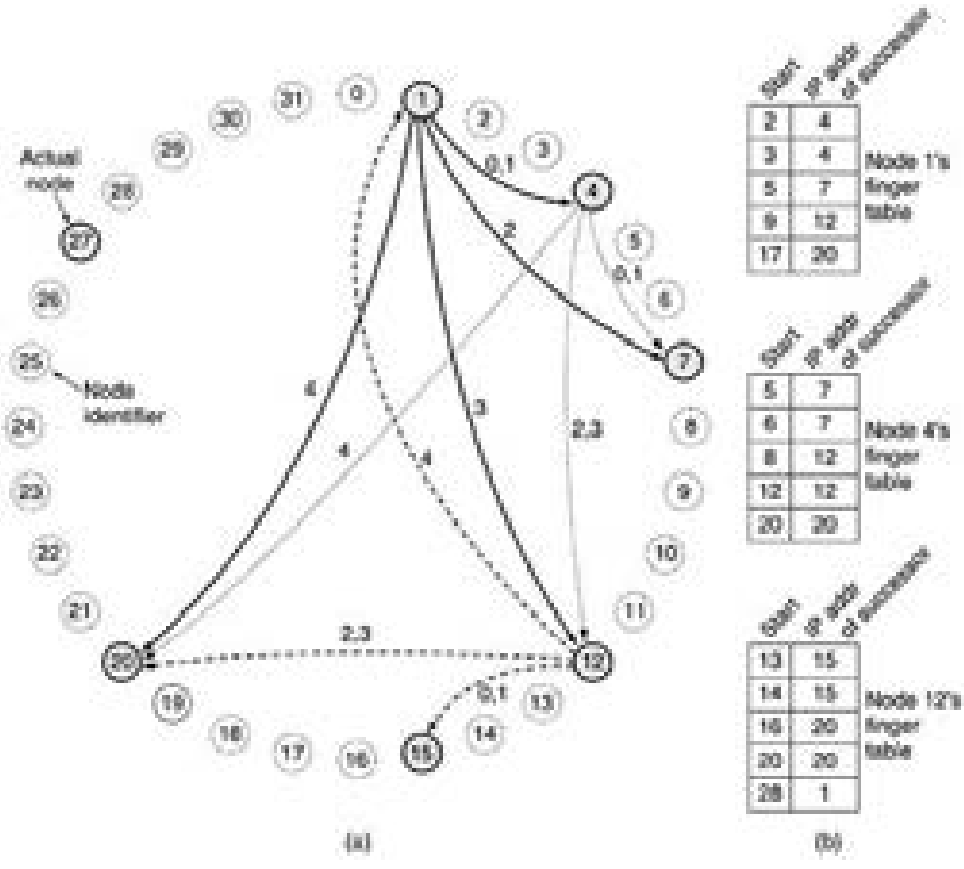


Figure 13-10. (a) A set of 32 node identifiers arranged in a circle. The shaded ones correspond to actual machines. The arcs show the fingers from nodes 1, 4, and 12. The labels on the arcs are the table indices. (b) Examples of the finger tables.

Let us now define the function successor(k) as the node identifier of the first actual node following k around the circle clockwise. For example, successor (6) = 7, successor (8) = 12, and successor (22) = 27.

The names of the records (song names, ancestors' names, and so on) are also hashed with hash (i.e., SHA-1) to generate a 160-bit number, called the key. Thus, to convert name (the ASCII name of the record) to its key, we use key = hash(name). This computation is just a local procedure call to hash. If a person holding a genealogical record for name wants to make it available to everyone, he first builds a tuple consisting of (name, my-IP-address) and then asks successor (hash(name)) to store the tuple.

If multiple records (at different nodes) exist for this name, their tuple will all be stored at the same node. In this way, the index is distributed over the nodes at random. For fault tolerance, p different hash functions could be used to store each tuple at p nodes, but we will not consider that further here.

If some user later wants to look up name, he hashes it to get key and then uses successor (key) to find the IP address of the node storing its index tuples. The first step is easy; the second one is not. To make it possible to find the IP address of the node corresponding to a certain key, each node must maintain certain administrative data structures.

One of these is the IP address of its successor node along the node identifier circle. For example, in Fig. 13-10, node 4's successor is 7 and node 7's successor is 12. Lookup can now proceed as follows. The requesting node sends a packet to its successor containing its IP address and the key it is looking for. The packet is propagated around the ring until it locates the successor to the node identifier being sought. That node checks to see if it has any information matching the key, and if so, returns it directly to the requesting node, whose IP address it has.

As a first optimization, each node could hold the IP addresses of both its successor and its predecessor, so that queries could be sent either clockwise or counterclockwise, depending on which path is thought to be shorter. For example, node 7 in Fig. 13-10 could go clockwise to find node identifier 10 but counterclockwise to find node identifier 3.

Even with two choices of direction, linearly searching all the nodes is very inefficient in a large peer-to-peer system since the mean number of nodes required per search is n/2. To greatly speed up the search, each node also maintains what Chord calls a finger table. The finger table has m entries, indexed by 0 through m - 1, each one pointing to a different actual node.

Each of the entries has two fields: start and the IP address of successor(start), as shown for three example nodes in Fig. 13-10(b). Note that each node stores the IP addresses of a relatively small number of nodes and that most of these are fairly close by in terms of node identifier. Using the finger table, the lookup of key at node k proceeds as follows. If key falls between k and successor (k), then the node holding information about key is successor (k) and the search terminates.

Otherwise, the finger table is searched to find the entry whose start field is the closest predecessor of key. A request is then sent directly to the IP address in that finger table entry to ask it to continue the search. Since it is closer to key but still below it, chances are good that it will be able to return the answer with only a small number of additional queries.

In fact, since every lookup halves the remaining distance to the target, it can be shown that the average number of lookups is $\log_2 n$.

As a first example, consider looking up key = 3 at node 1. Since node 1 knows that 3 lies between it and its successor, 4, the desired node is 4 and the search terminates, returning node 4's IP address.

As a second example, consider looking up key = 14 at node 1. Since 14 does not lie between 1 and 4, the finger table is consulted. The closest predecessor to 14 is 9, so the request is forwarded to the IP address of 9's entry, namely, that of node 12. Node 12 sees that 14 falls between it and its successor (15), so it returns the IP address of node 15.

As a third example, consider looking up key = 16 at node 1. Again a query is sent to node 12, but this time node 12 does not know the answer itself. It looks for the node most closely preceding 16 and finds 14, which yields the IP address of node 15. A query is then sent there. Node 15 observes that 16 lies between it and its successor (20), so it returns the IP address of 20 to the caller, which works its way back to node 1. Since nodes join and leave all the time, Chord needs a way to handle these operations.

We assume that when the system began operation it was small enough that the nodes could just exchange information directly to build the first circle and finger tables. After that an automated procedure is needed, as follows. When a new node, r, wants to join, it must contact some existing node and ask it to look up the IP address of successor (r) for it. The new node then asks successor (r) for its predecessor.

The new node then asks both of these to insert r in between them in the circle. For example, if 24 in Fig. 13-10 wants to join, it asks any node to look up successor (24), which is 27. Then it asks 27 for its predecessor (20). After it tells both of those about its existence, 20 uses 24 as its successor and 27 uses 24 as its predecessor. In addition, node 27 hands over those keys in the range 21–24, which now belong to 24. At this point, 24 is fully inserted.

However, many finger tables are now wrong. To correct them, every node runs a background process that periodically re-computes each finger by calling successor. When one of these queries hits a new node, the corresponding finger entry is updated. When a node leaves gracefully, it hands its keys over to its successor and informs its predecessor of its departure so the predecessor can link to the departing node's successor.

When a node crashes, a problem arises because its predecessor no longer has a valid successor. To alleviate this problem, each node keeps track not only of its direct successor but also its s direct successors, to allow it to skip over up to s - 1 consecutive failed nodes and reconnect the circle. Chord has been used to construct a distributed file system and other applications, and research is ongoing.

13.6 Summary

1. The routing path is divided into regions for the purpose of hierarchical routing which spans a large network.

2. In the application where data to be delivered to all the hosts broadcast routing is used.

3. Adhoc on Demand routing algorithms is used in the mobile Adhoc networks.

4. In a network where large numbers of people, usually with permanent wired connections to the Internet, are in contact to share resources use peer-to-peer network routing algorithms.

Lesson 14

Congestion Control Algorithms

Contents

14.0 Aim

One of the important performance metrics which determines the speed of the communication in the networked environment is the route traffic and data congestion. The following lesson details about the various sources of congestion and the algorithms to control congestion.

14.1 Introduction

When too many packets are present in (a part of) the subnet, performance degrades. This situation is called congestion. Fig.14-1 depicts the symptom. When the number of packets dumped into the subnet by the hosts is within its carrying capacity, they are all delivered (except for a few that are afflicted with transmission errors) and the number delivered is proportional to the number sent.

However, as traffic increases too far, the routers are no longer able to cope and they begin losing packets. This tends to make matters worse. At very high trafffic, performance collapses completely and almost no packets are delivered.

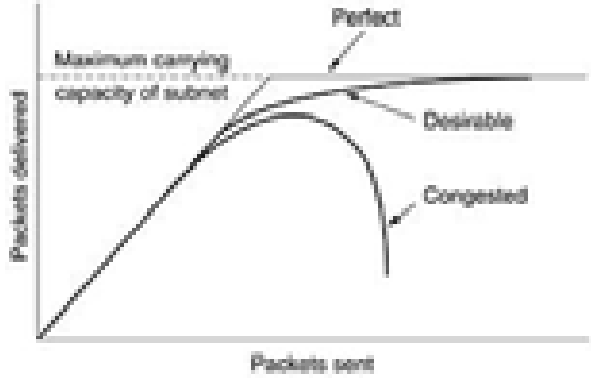


Figure 14-1. When too much traffic is offered, congestion sets in and performance degrades sharply.

Congestion can be brought on by several factors.

- ✓ If all of a sudden, streams of packets begin arriving on three or four input lines and all need the same output line, a queue will build up.
- ✓ If there is insufficient memory to hold all of them, packets will be lost. Adding more memory may help up to a point, but Nagle (1987) discovered that if routers have an infinite amount of memory, congestion gets worse, not better, because by the time packets get to the front of the queue, they have already timed out (repeatedly) and duplicates have been sent. All these packets will be dutifully forwarded to the next router, increasing the load all the way to the destination.
- ✓ Slow processors can also cause congestion. If the routers' CPUs are slow at performing the bookkeeping tasks required of them (queueing buffers, updating tables, etc.), queues can build up, even though there is excess line capacity.
- ✓ Similarly, low-bandwidth lines can also cause congestion. Upgrading the lines but not changing the processors, or vice versa, often helps a little, but frequently just shifts the bottleneck. Also, upgrading part, but not all, of the system, often just moves the bottleneck somewhere else. The real problem is frequently a mismatch between parts of the system. This problem will persist until all the components are in balance.

It is worth explicitly pointing out the difference between congestion control and flow control, as the relationship is subtle.

- Congestion control has to do with making sure the subnet is able to carry the offered traffic. It is a global issue, involving the behavior of all the hosts, all the routers, the store-and-forwarding processing within the routers, and all the other factors that tend to diminish the carrying capacity of the subnet.
- Flow control, in contrast, relates to the point-to-point traffic between a given sender and a given receiver. Its job is to make sure that a fast sender cannot continually transmit data faster than the receiver is able to absorb it. Flow control frequently involves some direct feedback from the receiver to the sender to tell the sender how things are doing at the other end.

14.2 General Principles of Congestion Control

Many problems in complex systems, such as computer networks, can be viewed from a control theory point of view. This approach leads to dividing all solutions into two groups:

1. Open Loop
2. Closed Loop

Open Loop

- ✓ Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place.
- ✓ Once the system is up and running, midcourse corrections are not made.
- ✓ Tools for doing open-loop control include deciding when to accept new traffic, deciding when to discard packets and which ones, and making scheduling decisions at various points in the network.

- ✓ All of these have in common the fact that they make decisions without regard to the current state of the network.

Closed Loop

- ✓ Closed loop solutions are based on the concept of a feedback loop.
- ✓ This approach has three parts when applied to congestion control:
    1. Monitor the system to detect when and where congestion occurs.
    2. Pass this information to places where action can be taken.
    3. Adjust system operation to correct the problem

Monitor the system to detect when and where congestion occurs

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.

Pass this information to places where action can be taken

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. However, other possibilities also exist. For example, a bit or field can be reserved in every packet for routers to fill in whenever congestion gets above some threshold level.

When a router detects this congested state, it fills in the field in all outgoing packets, to warn the neighbors. Still another approach is to have hosts or routers periodically send probe packets out to explicitly ask about congestion. This information can then be used to route traffic around problem areas. Some radio stations have helicopters flying around their cities to report on road congestion to make it possible for their mobile listeners to route their packets (cars) around hot spots.

In all feedback schemes, the hope is that knowledge of congestion will cause the hosts to take appropriate action to reduce the congestion. For a scheme to work correctly, the time scale must be adjusted carefully. If every time two packets arrive in a row, a router yells STOP and every time a router is idle for 20 μsec, it yells GO, the system will oscillate wildly and never converge.

On the other hand, if it waits 30 minutes to make sure before saying anything, the congestion control mechanism will react too sluggishly to be of any real use. To work well, some kind of averaging is needed, but getting the time constant right is a nontrivial matter.

Many congestion control algorithms are known. To provide a way to organize them in a sensible way, Yang and Reddy (1995) have developed taxonomy for congestion control algorithms. They begin by dividing all algorithms into open loop or closed loop, as described above. They further divide the open loop algorithms into ones that act at the source versus ones

that act at the destination. The closed loop algorithms are also divided into two subcategories: explicit feedback versus implicit feedback.

In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithms, the source deduces the existence of congestion by making local observations, such as the time needed for acknowledgements to come back.

Some of these methods can best be applied to virtual circuits. For subnets that use virtual circuits internally, these methods can be used at the network layer. For datagram subnets, they can nevertheless sometimes be used on transport layer connections.

14.3 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. In Fig.14-2 we see different data link, network, and transport policies that can affect congestion (Jain, 1990).

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

Figure 14-2. Policies that affect congestion.

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 n. 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 datagram affects congestion since many congestion control algorithms work only with virtual-circuit subnets.

Packet queuing 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.

14.4 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.

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. An alternative approach is to allow new virtual circuits but carefully route all new virtual circuits around problem areas. For example, consider the subnet of Fig.14-3(a), in which two routers are congested, as indicated.

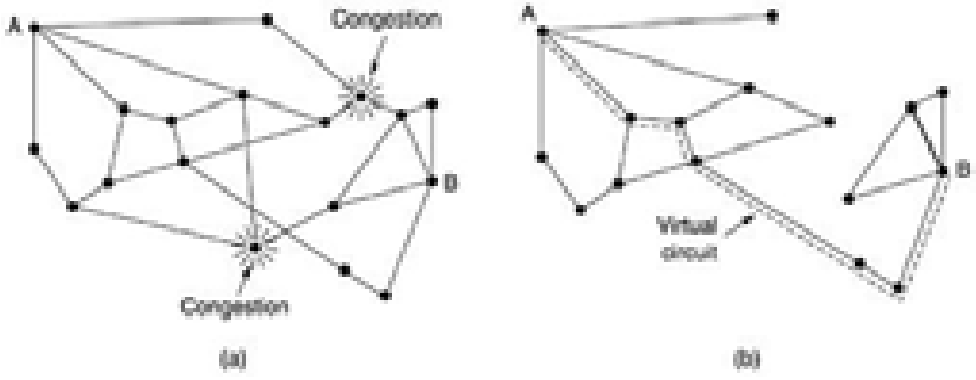


Figure 14-3. (a) A congested subnet. (b) A redrawn subnet that eliminates the congestion. A virtual circuit from A to B is also shown.

Suppose that a host attached to router A wants to set up a connection to a host attached to router B. Normally, this connection would pass through one of the congested routers. To avoid this situation, we can redraw the subnet as shown in Fig.14-3(b), omitting the congested routers and all of their lines.

The dashed line shows a possible route for the virtual circuit that avoids the congested routers. Another strategy relating to virtual circuits is to negotiate an agreement between the host and subnet when a virtual circuit is set up. This agreement normally specifies the volume and shape of the traffic, quality of service required, and other parameters. To keep its part of the agreement, the subnet will typically reserve resources along the path when the circuit is set up.

These resources can include table and buffer space in the routers and bandwidth on the lines. In this way, congestion is unlikely to occur on the new virtual circuits because all the necessary resources are guaranteed to be available. This kind of reservation can be done all the time as standard operating procedure or only when the subnet is congested. A disadvantage of doing it all the time is that it tends to waste resources.

If six virtual circuits that might use 1 Mbps all pass through the same physical 6-Mbps line, the line has to be marked as full, even though it may rarely happen that all six virtual circuits are transmitting full blast at the same time. Consequently, the price of the congestion control is unused (i.e., wasted) bandwidth in the normal case.

14.5 Congestion Control in Datagram Subnets

Some approaches that can be used in datagram subnets (and also in virtual-circuit subnets) are discussed below. 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 action taken can be one of several alternatives, which we will now discuss.

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

The previous congestion control algorithm is fairly subtle. It uses a roundabout means to tell the source to slow down. 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.

When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination by X percent. Since other packets aimed at the same destination are probably already under way and will generate yet more choke packets, the host should ignore choke packets referring to that destination for a fixed time interval.

After that period has expired, the host listens for more choke packets for another interval. If one arrives, the line is still congested, so the host reduces the flow still more and begins ignoring choke packets again. If no choke packets arrive during the listening period, the host may increase the flow again. The feedback implicit in this protocol can help prevent congestion yet not throttle any flow unless trouble occurs.

Hosts can reduce traffic by adjusting their policy parameters, for example, their window size. Typically, the first choke packet causes the data rate to be reduced to 0.50 of its previous rate, the next one causes a reduction to 0.25, and so on. Increases are done in smaller increments to prevent congestion from reoccurring quickly.

Several variations on this congestion control algorithm have been proposed. For one, the routers can maintain several thresholds. Depending on which threshold has been crossed, the choke packet can contain a mild warning, a stern warning, or an ultimatum. Another variation is to use queue lengths or buffer utilization instead of line utilization as the trigger signal. The same exponential weighting can be used with this metric as with u, of course.

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 in Fig.14-4) that is sending traffic to a host in New York (router D in Fig.14-4) 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 Fig.14-4(a). 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 in Fig.14-4(a) will the New York router notice a slower flow.

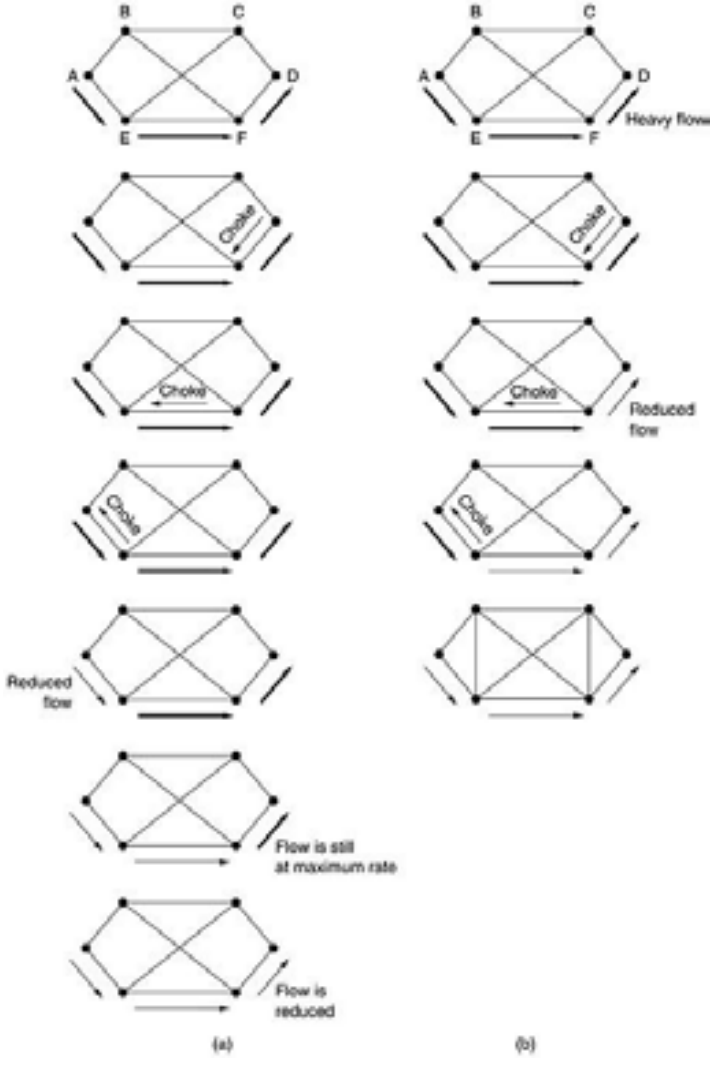


Figure 14-4. (a) A choke packet that affects only the source. (b) A choke packet that affects each hop it passes through.

An alternative approach is to have the choke packet take effect at every hop it passes through, as shown in the sequence of Fig.14-4(b). Here, as soon as the choke packet reaches F, F is required to reduce the flow to D. Doing so will require F to devote more buffers to the flow, since the source is still sending away at full blast, but it gives D immediate relief, like a headache remedy in a television commercial.

In the next step, the choke packet reaches E, which tells E to reduce the flow to F. This action puts a greater demand on E's buffers but gives F immediate relief. Finally, the choke packet reaches A and the flow genuinely slows down.

The net effect of this hop-by-hop scheme is to provide quick relief at the point of congestion at the price of using up more buffers upstream. In this way, congestion can be nipped in the bud without losing any packets.

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. A step above this in intelligence requires cooperation from the senders. For many applications, some packets are more important than others.

For example, certain algorithms for compressing video periodically transmit an entire frame and then send subsequent frames as differences from the last full frame. In this case, dropping a packet that is part of a difference is preferable to dropping one that is part of a full frame. As another example, consider transmitting a document containing ASCII text and pictures. Losing a line of pixels in some image is far less damaging than losing a line of readable text.

To implement an intelligent discard policy, applications must mark their packets in priority classes to indicate how important they are. If they do this, then when packets have to be discarded, routers can first drop packets from the lowest class, then the next lowest class, and so on. Of course, unless there is some significant incentive to mark packets as anything other than VERY IMPORTANT— NEVER, EVER DISCARD, nobody will do it.

The incentive might be in the form of money, with the low-priority packets being cheaper to send than the high-priority ones. Alternatively, senders might be allowed to send high-priority packets under conditions of light load, but as the load increased they would be discarded, thus encouraging the users to stop sending them. Another option is to allow hosts to exceed the limits specified in the agreement negotiated when the virtual circuit was set up (e.g., use a higher bandwidth than allowed), but subject to the condition that all excess traffic be marked as low priority.

Such a strategy is actually not a bad idea, because it makes more efficient use of idle resources, allowing hosts to use them as long as nobody else is interested, but without establishing a right to them when times get tough.

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.

By having routers drop packets before the situation has become hopeless (hence the "early" in the name), the idea is that there is time for action to be taken before it is too late. To determine when to start discarding, routers maintain a running average of their queue lengths.

When the average queue length on some line exceeds a threshold, the line is said to be congested and action is taken. Since the router probably cannot tell which source is causing most of the trouble, picking a packet at random from the queue that triggered the action is probably as good as it can do. How should the router tell the source about the problem? One way is to send it a choke packet, as we have described.

A problem with that approach is that it puts even more load on the already congested network. A different strategy is to just discard the selected packet and not report it. The source will eventually notice the lack of acknowledgement and take action. Since it knows that lost packets are generally caused by congestion and discards, it will respond by slowing down instead of trying harder.

This implicit form of feedback only works when sources respond to lost packets by slowing down their transmission rate. In wireless networks, where most losses are due to noise on the air link, this approach cannot be used.

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.

Jitter is illustrated in Fig.14-5. 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.

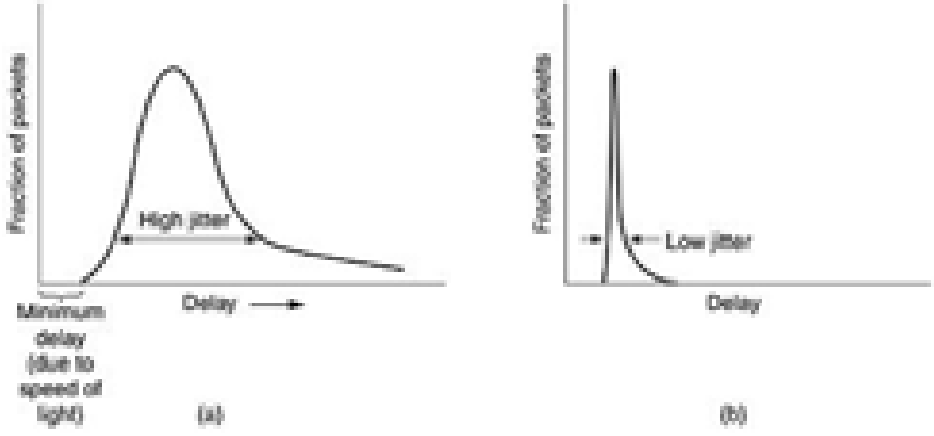


Figure 14-5. (a) High jitter. (b) Low jitter.

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, packets that are ahead of schedule get slowed down and packets that are behind schedule get speeded up, in both cases reducing the amount of jitter. In some applications, such as video on demand, jitter can be eliminated by buffering at the receiver and then fetching data for display from the buffer instead of from the network in real time.

However, for other applications, especially those that require real-time interaction between people such as Internet telephony and videoconferencing, the delay inherent in buffering is not acceptable.

14.6 Summary

1. Congestion is to the presence of more number of packets in the subnet.
2. Congestion prevention policies may be adapted in virtual circuits and datagram subnets.

Lesson 15

TRANSPORT LAYER

Contents

15.0 Aim

The reliable means of communication with virtual circuit connectivity comes with the TCP protocol. The various concepts involving the connection establishment, release, flow control and buffering are explained in this lesson.

15.1 Introduction

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. However, significant differences between the two also exist. These differences are due to major dissimilarities between the environments in which the two protocols operate, as shown in Fig. 15-1.

15.2 Elements of Transport Protocols

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.

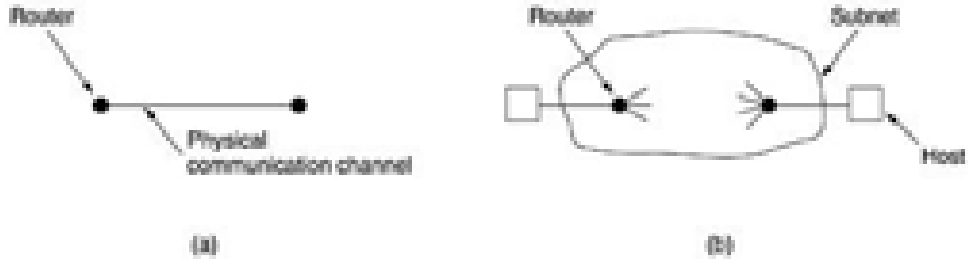


Fig. 15-1 (a) Environment of the data link layer. (b) Environment of the transport layer.

For one thing, in the data link layer, it is not necessary for a router to specify which router it wants to talk to—each outgoing line uniquely specifies a

particular router. In the transport layer, explicit addressing of destinations is required.

For another thing, the process of establishing a connection over the wire of Fig. 15-1(a) is simple: the other end is always there (unless it has crashed, in which case it is not there). Either way, there is not much to do. In the transport layer, initial connection establishment is more complicated, as we will see.

Another, exceedingly annoying, difference between the data link layer and the transport layer is the potential existence of storage capacity in the subnet. When a router sends a frame, it may arrive or be lost, but it cannot bounce around for a while, go into hiding in a far corner of the world, and then suddenly emerge at an inopportune moment 30 sec later.

If the subnet uses datagrams and adaptive routing inside, there is a non-negligible probability that a packet may be stored for a number of seconds and then delivered later. The consequences of the subnet's ability to store packets can sometimes be disastrous and can require the use of special protocols.

A final difference between the data link and transport layers is one of amount rather than of kind. Buffering and flow control are needed in both layers, but the presence of a large and dynamically varying number of connections in the transport layer may require a different approach than we used in the data link layer.

15.2.1 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. 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.

Fig. 15-2 illustrates 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.

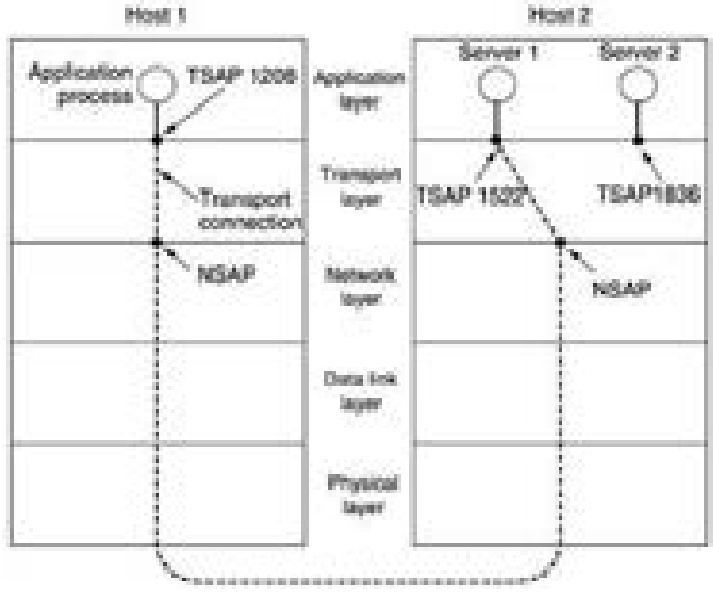


Fig. 15-2. TSAPs, NSAPs, and transport connections.

- A possible scenario for a transport connection is as follows.
  1. 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.
  2. 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.
  3. The application process then sends over a request for the time.
  4. The time server process responds with the current time.
  5. 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.

While stable TSAP addresses work for a small number of key services that never change (e.g. the Web server), user processes, in general, often want to talk to other user processes that only exist for a short time and do not have a TSAP address that is known in advance.

One such scheme is shown in Fig. 15-3 in a simplified form. It is known as the initial connection protocol. Instead of every conceivable server listening at a well-known TSAP, each machine that wishes to offer services to remote users has a special process server that acts as a proxy for less heavily used servers.

It listens to a set of ports at the same time, waiting for a connection request. Potential users of a service begin by doing a CONNECT request, specifying the TSAP address of the service they want. If no server is waiting for them, they get a connection to the process server, as shown in Fig. 15-3(a).

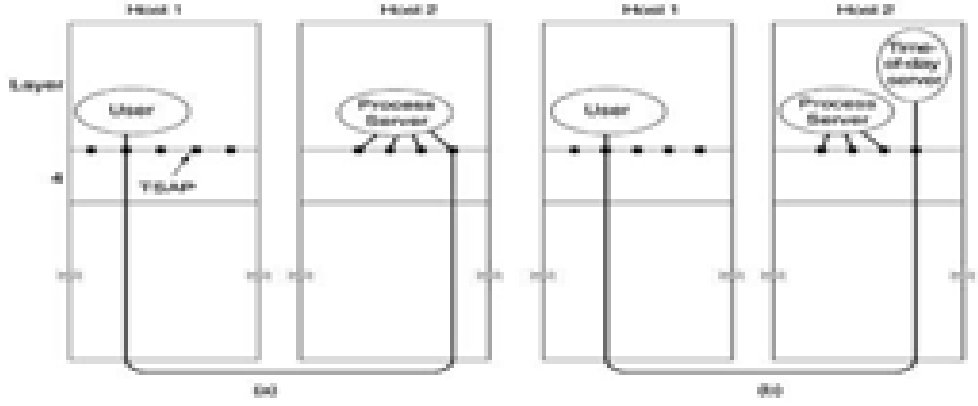


Fig.15-3. How a user process in host 1 establishes a connection with a time-of-day server in host 2.

After it gets the incoming request, the process server spawns the requested server, allowing it to inherit the existing connection with the user. The new server then does the requested work, while the process server goes back to listening for new requests, as shown in Fig. 15-3(b). While the initial connection protocol works fine for those servers that can be created as they are needed, there are many situations in which services do exist independently of the process server.

A file server, for example, needs to run on special hardware (a machine with a disk) and cannot just be created on-the-fly when someone wants to talk to it. To handle this situation, an alternative scheme is often used. In this model, there exists a special process called a name server or sometimes a directory server. To find the TSAP address corresponding to a given service name, such as "time of day," a user sets up a connection to the name server (which listens to a well-known TSAP).

The user then sends a message specifying the service name, and the name server sends back the TSAP address. Then the user releases the connection with the name server and establishes a new one with the desired service.

In this model, when a new service is created, it must register itself with the name server, giving both its service name (typically, an ASCII string) and its TSAP. The name server records this information in its internal database so that when queries come in later, it will know the answers.

The function of the name server is analogous to the directory assistance operator in the telephone system—it provides a mapping of names onto numbers. Just as in the telephone system, it is essential that the address of the

well-known TSAP used by the name server (or the process server in the initial connection protocol) is indeed well known.

15.2.2 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.
- The worst possible nightmare is as follows.

A user establishes a connection with a bank, sends messages telling the bank to transfer a large amount of money to the account of a not-entirely-trustworthy person, and then releases the connection. Unfortunately, each packet in the scenario is duplicated and stored in the subnet. After the connection has been released, all the packets pop out of the subnet and arrive at the destination in order, asking the bank to establish a new connection, transfer money (again), and release the connection. The bank has no way of telling that these are duplicates. It must assume that this is a second, independent transaction, and transfers the money again. For the remainder of this section we will study the problem of delayed duplicates, with special emphasis on algorithms for establishing connections in a reliable way, so that nightmares like the one above cannot happen.

The crux of the problem is the existence of delayed duplicates. It can be attacked in various ways, none of them very satisfactory. One way is to use throw-away transport addresses. In this approach, each time a transport address is needed, a new one is generated. When a connection is released, the address is discarded and never used again. This strategy makes the process server model of Fig. 15-3 impossible.

Another possibility is to give each connection a connection identifier (i.e., a sequence number incremented for each connection established) chosen by the initiating party and put in each TPDU, including the one requesting the connection.

After each connection is released, each transport entity could update a table listing obsolete connections as (peer transport entity, connection identifier) pairs. Whenever a connection request comes in, it could be checked against the table, to see if it belonged to a previously-released connection.

Unfortunately, this scheme has a basic flaw: it requires each transport entity to maintain a certain amount of history information indefinitely. Instead, we need to take a different track. Rather than allowing packets to live forever within the subnet, we must devise a mechanism to kill off aged packets that are still hobbling about. If we can ensure that no packet lives longer than some

known time, the problem becomes somewhat more manageable. Packet lifetime can be restricted to a known maximum using one (or more) of the following techniques:

1. Restricted subnet design.
2. Putting a hop counter in each packet.
3. Timestamping 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.

In practice, we will need to guarantee not only that a packet is dead, but also that all acknowledgements to it are also dead, so we will now introduce T, which is some small multiple of the true maximum packet lifetime. The multiple is protocol dependent and simply has the effect of making T longer. If we wait a time T after a packet has been sent, we can be sure that all traces of it are now gone and that neither it nor its acknowledgements will suddenly appear out of the blue to complicate matters.

To get around the problem of a machine losing all memory of where it was after a crash, Tomlinson proposed equipping each host with a time-of-day clock. The clocks at different hosts need not be synchronized. Each clock is assumed to take the form of a binary counter that increments itself at uniform intervals.

Furthermore, the number of bits in the counter must equal or exceed the number of bits in the sequence numbers. Last, and most important, the clock is assumed to continue running even if the host goes down. The basic idea is to ensure that two identically numbered TPDUs are never outstanding at the same time.

When a connection is set up, the low-order k bits of the clock are used as the initial sequence number (also k bits). Thus, unlike our old protocols each connection starts numbering its TPDUs with a different initial sequence number. The sequence space should be so large that by the time sequence numbers wrap around, old TPDUs with the same sequence number are long gone. This linear relation between time and initial sequence numbers is shown in Fig. 15-4.

Once both transport entities have agreed on the initial sequence number, any sliding window protocol can be used for data flow control. In reality, the initial sequence number curve (shown by the heavy line) is not linear, but a staircase, since the clock advances in discrete steps. For simplicity we will ignore this detail.

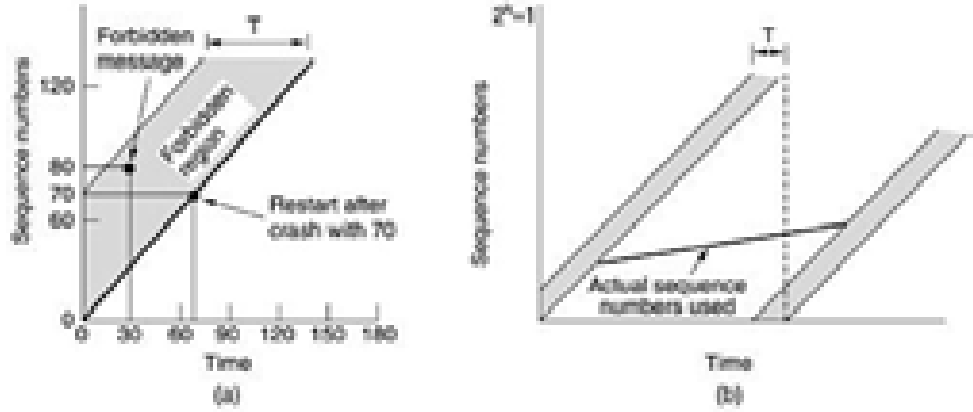


Figure 15-4 (a) TPDUs may not enter the forbidden region. (b) The resynchronization problem.

A problem occurs when a host crashes. When it comes up again, its transport entity does not know where it was in the sequence space. One solution is to require transport entities to be idle for T sec after a recovery to let all old TPDUs die off. However, in a complex internetwork, T may be large, so this strategy is unattractive.

To avoid requiring T sec of dead time after a crash, it is necessary to introduce a new restriction on the use of sequence numbers. We can best see the need for this restriction by means of an example. Let T, the maximum packet lifetime, be 60 sec and let the clock tick once per second. As shown by the heavy line in Fig. 15-4(a), the initial sequence number for a connection opened at time x will be x.

Imagine that at t = 30 sec, an ordinary data TPDU being sent on (a previously opened) connection 5 is given sequence number 80. Call this TPDU X. Immediately after sending TPDU X, the host crashes and then quickly restarts. At t = 60, it begins reopening connections 0 through 4. At t = 70, it reopens connection 5, using initial sequence number 70 as required. Within the next 15 sec it sends data TPDUs 70 through 80. Thus, at t = 85 a new TPDU with sequence number 80 and connection 5 has been injected into the subnet.

Unfortunately, TPDU X still exists. If it should arrive at the receiver before the new TPDU 80, TPDU X will be accepted and the correct TPDU 80 will be rejected as a duplicate. To prevent such problems, we must prevent sequence numbers from being used (i.e., assigned to new TPDUs) for a time T before their potential use as initial sequence numbers.

The illegal combinations of time and sequence number are shown as the forbidden region in Fig. 15-4(a). Before sending any TPDU on any connection, the transport entity must read the clock and check to see that it is not in the forbidden region.

The protocol can get itself into trouble in two distinct ways. If a host sends too much data too fast on a newly-opened connection, the actual sequence number versus time curve may rise more steeply than the initial sequence number versus time curve. This means that the maximum data rate on any connection is one TPDU per clock tick.

It also means that the transport entity must wait until the clock ticks before opening a new connection after a crash restart, lest the same number be used twice. Both of these points argue in favor of a short clock tick (a few μsec

or less). Unfortunately, entering the forbidden region from underneath by sending too fast is not the only way to get into trouble.

From Fig. 15-4(b), we see that at any data rate less than the clock rate, the curve of actual sequence numbers used versus time will eventually run into the forbidden region from the left. The greater the slope of the actual sequence number curve, the longer this event will be delayed.

As we stated above, just before sending every TPDU, the transport entity must check to see if it is about to enter the forbidden region, and if so, either delay the TPDU for T sec or resynchronize the sequence numbers. The clock-based method solves the delayed duplicate problem for data TPDUs, but for this method to be useful, a connection must first be established.

Since control TPDUs may also be delayed, there is a potential problem in getting both sides to agree on the initial sequence number. Suppose, for example, that connections are established by having host 1 send a CONNECTION REQUEST TPDU containing the proposed initial sequence number and destination port number to a remote peer, host 2.

The receiver, host 2, then acknowledges this request by sending a CONNECTION ACCEPTED TPDU back. If the CONNECTION REQUEST TPDU is lost but a delayed duplicate CONNECTION REQUEST suddenly shows up at host 2, the connection will be established incorrectly.

To solve this problem, Tomlinson (1975) introduced the three-way handshake.

- This establishment protocol does not require both sides to begin sending with the same sequence number, so it can be used with synchronization methods other than the global clock method.
- The normal setup procedure when host 1 initiates is shown in Fig. 15-5. Host 1 chooses a sequence number, x, and sends a CONNECTION REQUEST TPDU containing it to host 2.
- Host 2 replies with an ACK TPDU acknowledging x and announcing its own initial sequence number, y. Finally, host 1 acknowledges host 2's choice of an initial sequence number in the first data TPDU that it sends.

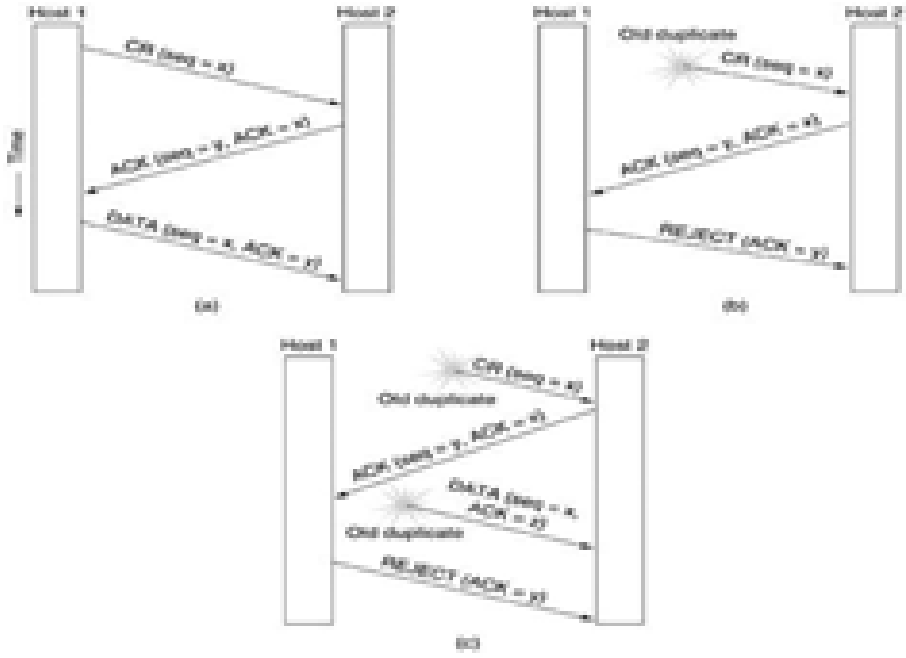


Figure 15-5 Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. (a) Normal operation. (b) Old duplicate CONNECTION REQUEST appearing out of nowhere. (c) Duplicate CONNECTION REQUEST and duplicate ACK.

Now let us see how the three-way handshake works in the presence of delayed duplicate control TPDUs.

In Fig. 15-5, the first TPDU is a delayed duplicate CONNECTION REQUEST from an old connection. This TPDU arrives at host 2 without host 1's knowledge. Host 2 reacts to this TPDU by sending host 1 and ACK TPDU, in effect asking for verification that host 1 was indeed trying to set up a new connection.

When host 1 rejects host 2's attempt to establish a connection, host 2 realizes that it was tricked by a delayed duplicate and abandons the connection. In this way, a delayed duplicate does no damage.

The worst case is when both a delayed CONNECTION REQUEST and an ACK are floating around in the subnet. This case is shown in Fig. 15-5(c). As in the previous example, host 2 gets a delayed CONNECTION REQUEST and replies to it. At this point it is crucial to realize that host 2 has proposed using y as the initial sequence number for host 2 to host 1 traffic, knowing full well that no TPDUs containing sequence number y or acknowledgements to y are still in existence.

When the second delayed TPDU arrives at host 2, the fact that z has been acknowledged rather than y tells host 2 that this, too, is an old duplicate. The important thing to realize here is that there is no combination of old TPDUs that can cause the protocol to fail and have a connection set up by accident when no one wants it.

15.2.3 Connection Release

Releasing a connection is easier than establishing one. There are more pitfalls than one might expect. There are two styles of terminating a connection:

1. Asymmetric Release
2. Symmetric Release

Asymmetric Release

- Asymmetric release is the way the telephone system works: when one party hangs up, the connection is broken.
- Asymmetric release is abrupt and may result in data loss.
- Consider the scenario of Fig. 15-6 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.

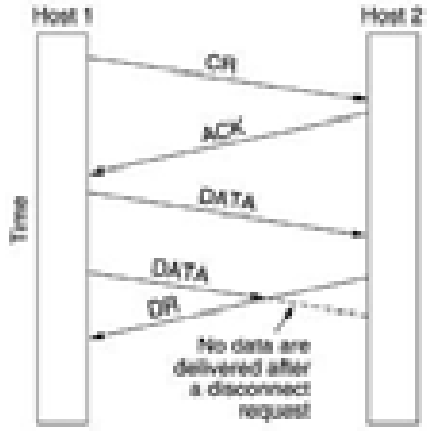


Figure 15-6. Abrupt disconnection with loss of data

- Clearly, a more sophisticated release protocol is needed to avoid data loss.
- One way is to use symmetric release, in which each direction is released independently of the other one.
- Here, a host can continue to receive data even after it has sent a DISCONNECT TPDU.

Symmetric Release

- Symmetric release treats the connection as two separate unidirectional connections and requires each one to be released separately.
- Symmetric release does the job when each process has a fixed amount of data to send and clearly knows when it has sent it.

In other situations, determining that all the work has been done and the connection should be terminated is not so obvious. One can envision a protocol in which host 1 says: I am done. Are you done too? If host 2 responds: I am done too. Goodbye, the connection can be safely released.

Unfortunately, this protocol does not always work. There is a famous problem that illustrates this issue. It is called the two-army problem. Imagine that a white army is encamped in a valley, as shown in Fig. 15-7 On both of the surrounding hillsides are blue armies. The white army is larger than either of the blue armies alone, but together the blue armies are larger than the white army.

If either blue army attacks by itself, it will be defeated, but if the two blue armies attack simultaneously, they will be victorious.

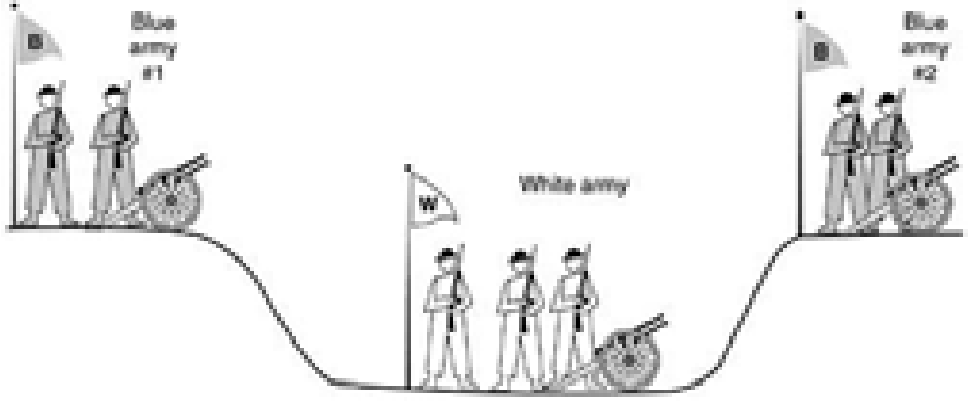


Figure 15-7 The two-army problem.

The blue armies want to synchronize their attacks. However, their only communication medium is to send messengers on foot down into the valley, where they might be captured and the message lost (i.e., they have to use an unreliable communication channel). The question is: Does a protocol exist that allows the blue armies to win?

Suppose that the commander of blue army #1 sends a message reading: "I propose we attack at dawn on March 29. How about it?" Now suppose that the message arrives, the commander of blue army #2 agrees, and his reply gets safely back to blue army #1. Will the attack happen? Probably not, because commander #2 does not know if his reply got through. If it did not, blue army #1 will not attack, so it would be foolish for him to charge into battle. Now let us improve the protocol by making it a three-way handshake. The initiator of the original proposal must acknowledge the response.

Assuming no messages are lost, blue army #2 will get the acknowledgement, but the commander of blue army #1 will now hesitate. After all, he does not know if his acknowledgement got through, and if it did not, he knows that blue army #2 will not attack. We could now make a four-way handshake protocol, but that does not help either.

In fact, it can be proven that no protocol exists that works. Suppose that some protocol did exist. Either the last message of the protocol is essential or it is not. If it is not, remove it (and any other unessential messages) until we are left with a protocol in which every message is essential.

What happens if the final message does not get through? We just said that it was essential, so if it is lost, the attack does not take place. Since the sender of the final message can never be sure of its arrival, he will not risk attacking. Worse yet, the other blue army knows this, so it will not attack

either. To see the relevance of the two-army problem to releasing connections, just substitute "disconnect" for "attack."

If neither side is prepared to disconnect until it is convinced that the other side is prepared to disconnect too, the disconnection will never happen. In practice, one is usually prepared to take more risks when releasing connections than when attacking white armies, so the situation is not entirely hopeless. Fig. 15-8 illustrates four scenarios of releasing using a three-way handshake. While this protocol is not infallible, it is usually adequate.

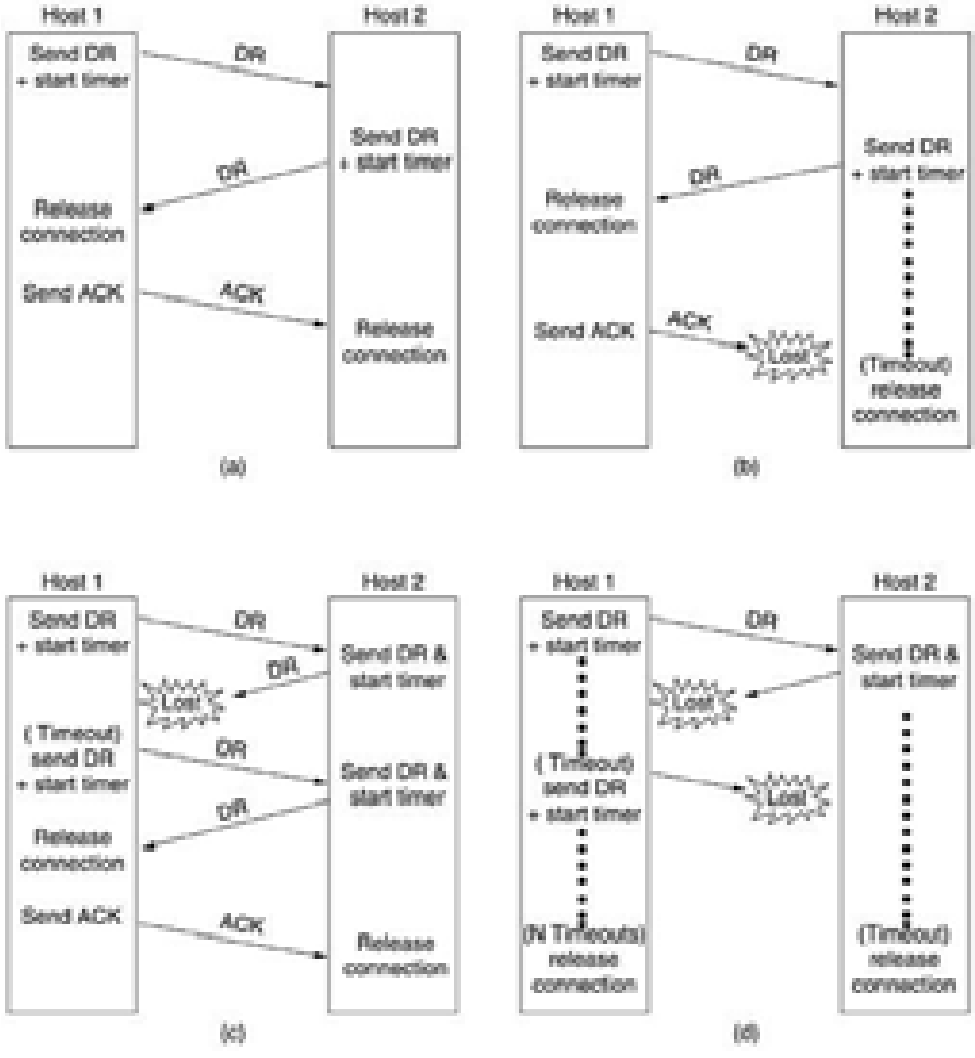


Figure 15-8. Four protocol scenarios for releasing a connection. (a) Normal case of three-way handshake. (b) Final ACK lost. (c) Response lost. (d) Response lost and subsequent DRs lost.

In Fig. 15-8(a), we see the normal case in which one of the users sends a DR (DISCONNECTION REQUEST) TPDU to initiate the connection release. When it arrives, the recipient sends back a DR TPDU, too, and starts a timer, just in case its DR is lost. When this DR arrives, the original sender sends back an ACK TPDU and releases the connection. Finally, when the ACK TPDU arrives, the receiver also releases the connection.

15.2.4 Flow Control and Buffering

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 the data link protocols of, frames were buffered at both the sending router and at the receiving router. 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.

In summary, if the network service is unreliable, the sender must buffer all TPDUs sent, just as in the data link layer. However, with reliable network service, other trade-offs become possible. In particular, if the sender knows that the receiver always has buffer space, it need not retain copies of the TPDUs it sends. However, if the receiver cannot guarantee that every incoming TPDU will be accepted, the sender will have to buffer anyway.

In the latter case, the sender cannot trust the network layer's acknowledgement, because the acknowledgement means only that the TPDU arrived, not that it was accepted. Even if the receiver has agreed to do the buffering, there still remains the question of the buffer size. If most TPDUs are nearly the same size, it is natural to organize the buffers as a pool of identically-sized buffers, with one TPDU per buffer, as in Fig. 15-9(a), If the buffer size is chosen equal to the largest possible TPDU, space will be wasted whenever a short TPDU arrives. If the buffer size is chosen less than the maximum TPDU size, multiple buffers will be needed for long TPDUs, with the attendant complexity.

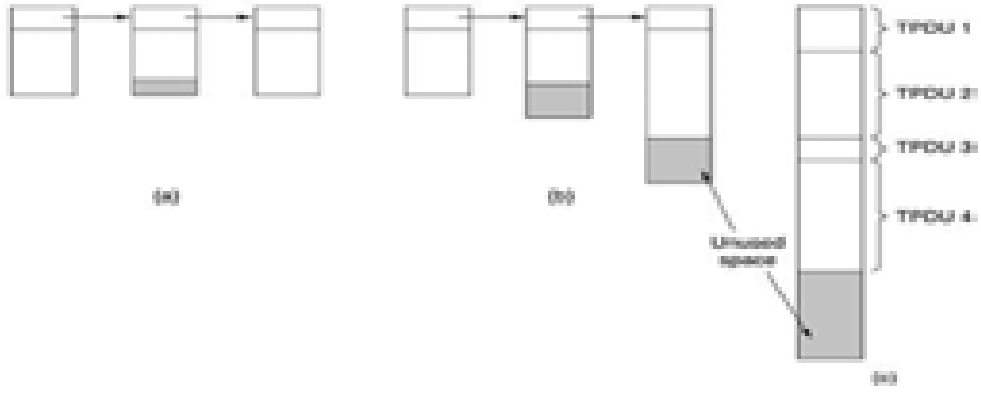


Figure 15-9. (a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular buffer per connection.

Another approach to the buffer size problem is to use variable-sized buffers, as in Fig. 15-9(b). The advantage here is better memory utilization, at the price of more complicated buffer management. A third possibility is to dedicate a single large circular buffer per connection, as in Fig. 15-9(c). This system also makes good use of memory, provided that all connections are heavily loaded, but is poor if some connections are lightly loaded.

The optimum trade-off between source buffering and destination buffering depends on the type of traffic carried by the connection. For low-bandwidth bursty traffic, such as that produced by an interactive terminal, it is better not to dedicate any buffers, but rather to acquire them dynamically at both ends. Since the sender cannot be sure the receiver will be able to acquire a buffer, the sender must retain a copy of the TPDU until it is acknowledged.

On the other hand, for file transfer and other high-bandwidth traffic, it is better if the receiver does dedicate a full window of buffers, to allow the data to flow at maximum speed. Thus, for low-bandwidth bursty traffic, it is better to buffer at the sender, and for high bandwidth smooth traffic, it is better to buffer at the receiver.

As connections are opened and closed and as the traffic pattern changes, the sender and receiver need to dynamically adjust their buffer allocations. Consequently, the transport protocol should allow a sending host to request buffer space at the other end. Buffers could be allocated per connection, or collectively, for all the connections running between the two hosts.

Alternatively, the receiver, knowing its buffer situation (but not knowing the offered traffic) could tell the sender "I have reserved X buffers for you." If the number of open connections should increase, it may be necessary for an allocation to be reduced, so the protocol should provide for this possibility. A reasonably general way to manage dynamic buffer allocation is to decouple the buffering from the acknowledgements, in contrast to the sliding window protocols.

Dynamic buffer management means, in effect, a variable-sized window. Initially, the sender requests a certain number of buffers, based on its perceived needs. The receiver then grants as many of these as it can afford. Every time the sender transmits a TPDU, it must decrement its allocation, stopping altogether when the allocation reaches zero. The receiver then separately piggybacks both acknowledgements and buffer allocations onto the reverse traffic.

Fig. 15-10 shows an example of how dynamic window management might work in a datagram subnet with 4-bit sequence numbers. Assume that buffer allocation information travels in separate TPDUs, as shown, and is not piggybacked onto reverse traffic. Initially, A wants eight buffers, but is granted only four of these. It then sends three TPDUs, of which the third is lost. TPDU 6 acknowledges receipt of all TPDUs up to and including sequence number 1, thus allowing A to release those buffers, and furthermore informs A that it has permission to send three more TPDUs starting beyond 1 (i.e., TPDUs 2, 3, and 4).

A knows that it has already sent number 2, so it thinks that it may send TPDUs 3 and 4, which it proceeds to do. At this point it is blocked and must wait for more buffer allocation. Timeout-induced retransmissions (line 9), however, may occur while blocked, since they use buffers that have already been allocated. In line 10, B acknowledges receipt of all TPDUs up to and including 4 but refuses to let A continue.

Such a situation is impossible with the fixed window protocols. The next TPDU from B to A allocates another buffer and allows A to continue.

Potential problems with buffer allocation schemes of this kind can arise in datagram networks if control TPDUs can get lost. Look at line 16. B has now allocated more buffers to A, but the allocation TPDU was lost. Since control TPDUs are not sequenced or timed out, A is now deadlocked.

To prevent this situation, each host should periodically send control TPDUs giving the acknowledgement and buffer status on each connection. That way, the deadlock will be broken, sooner or later. Until now we have tacitly assumed that the only limit imposed on the sender's data rate is the amount of buffer space available in the receiver.

As memory prices continue to fall dramatically, it may become feasible to equip hosts with so much memory that lack of buffers is rarely, if ever, a problem. When buffer space no longer limits the maximum flow, another bottleneck will appear: the carrying capacity of the subnet. If adjacent routers can exchange at most x packets/sec and there are k disjoint paths between a pair of hosts, there is no way that those hosts can exchange more than kx TPDUs/sec, no matter how much buffer space is available at each end.

Figure 15-10. Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (...) indicates a lost TPDU.

If the sender pushes too hard (i.e., sends more than kx TPDUs/sec), the subnet will become congested because it will be unable to deliver TPDUs as fast as they are coming in. What is needed is a mechanism based on the subnet's carrying capacity rather than on the receiver's buffering capacity. Clearly, the flow control mechanism must be applied at the sender to prevent it from having too many unacknowledged TPDUs outstanding at once. Belsnes (1975) proposed using a sliding window flow control scheme in which the sender dynamically adjusts the window size to match the network's carrying capacity.

If the network can handle c TPDUs/sec and the cycle time (including transmission, propagation, queueing, processing at the receiver, and return of the acknowledgement) is r, then the sender's window should be cr. With a window of this size the sender normally operates with the pipeline full. Any small decrease in network performance will cause it to block. In order to adjust the window size periodically, the sender could monitor both parameters and then compute the desired window size.

The carrying capacity can be determined by simply counting the number of TPDUs acknowledged during some time period and then dividing by the time period. During the measurement, the sender should send as fast as it can, to make sure that the network's carrying capacity, and not the low input rate, is the factor limiting the acknowledgement rate.

The time required for a transmitted TPDU to be acknowledged can be measured exactly and a running mean maintained. Since the network capacity available to any given flow varies in time, the window size should be adjusted frequently, to track changes in the carrying capacity.

15.2.5 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, is shown in Fig. 15-11(a). In this figure, four distinct transport connections all use the same network connection (e.g., IP address) to the remote host.

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.

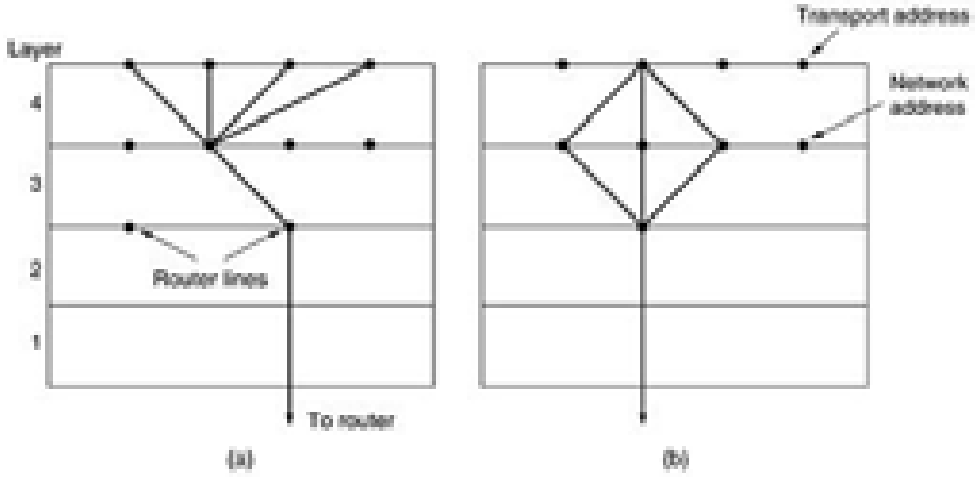


Figure 15-11. (a) Upward multiplexing. (b) Downward multiplexing.

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, as indicated in Fig. 15-11(b). 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.

15.2.6 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.

At first glance it would seem obvious: the client should retransmit only if and only if it has an unacknowledged TPDU outstanding (i.e., is in state S1) when it learns of the crash. However, a closer inspection reveals difficulties with this naive approach. Consider, for example, the situation in which the server's transport entity first sends an acknowledgement, and then, when the acknowledgement has been sent, writes to the application process.

Writing a TPDU onto the output stream and sending an acknowledgement are two distinct events that cannot be done simultaneously. If a crash occurs after the acknowledgement has been sent but before the write has been done, the client will receive the acknowledgement and thus be in state S0 when the crash recovery announcement arrives.

The client will therefore not retransmit, (incorrectly) thinking that the TPDU has arrived. This decision by the client leads to a missing TPDU. At this point you may be thinking: ''That problem can be solved easily. All you have to do is reprogram the transport entity to first do the write and then send the acknowledgement.'' Try again.

Imagine that the write has been done but the crash occurs before the acknowledgement can be sent. The client will be in state S1 and thus retransmit, leading to an undetected duplicate TPDU in the output stream to the server application process. No matter how the client and server are programmed, there are always situations where the protocol fails to recover properly.

The server can be programmed in one of two ways: acknowledge first or write first. The client can be programmed in one of four ways: always retransmit the last TPDU, never retransmit the last TPDU, retransmit only in state S0, or retransmit only in state S1. This gives eight combinations, but as we shall see, for each combination there is some set of events that makes the protocol fail.

Three events are possible at the server: sending an acknowledgement (A), writing to the output process (W), and crashing (C). The three events can occur in six different orderings: AC(W), AWC, C(AW), C(WA), WAC, and WC(A), where the parentheses are used to indicate that neither A nor W can follow C (i.e., once it has crashed, it has crashed).

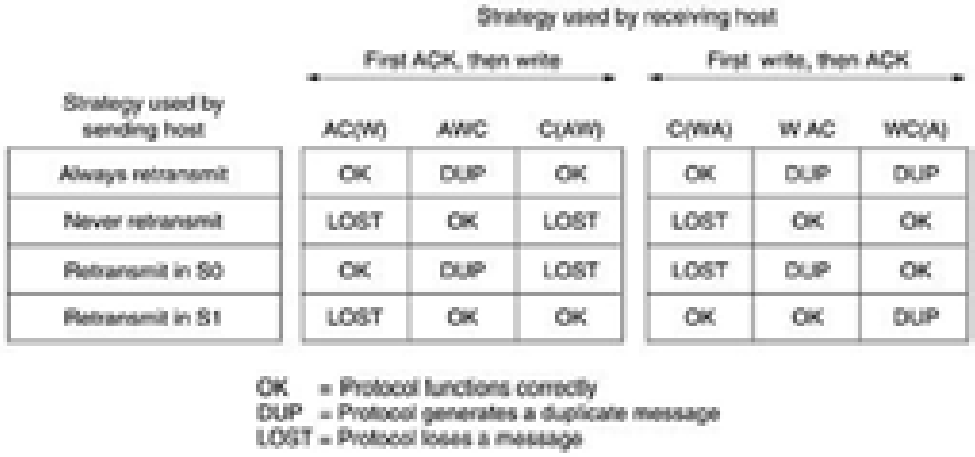


Figure 15-12. Different combinations of client and server strategy.

Fig. 15-12 shows all eight combinations of client and server strategy and the valid event sequences for each one. Notice that for each strategy there is some sequence of events that causes the protocol to fail. For example, if the client always retransmits, the AWC event will generate an undetected duplicate, even though the other two events work properly.

15.3 Summary

1. The transport layer is responsible for addressing, connection establishment, release and overflow buffer management.
2. Connection establishment is done through the request and acknowledgement packets.
3. Connection release is carried out by the finish packets.
4. Various types of buffers including Chained fixed-size buffers, Chained variable-sized buffers and One large circular buffer per connection.

Lesson 16

The Internet Transport Protocols: TCP

Contents

16.0 Aim

This lesson deals about internet protocol TCP. The various techniques for connection establishment and services by the TCP are detailed in this lesson.

16.1 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.

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 datagram containing TCP data arrive at a machine, they are given to the TCP entity, which reconstructs the original byte streams.

Datagram 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.

16.2 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. The socket calls are listed in Fig 16-1. 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 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.

The list of well-known ports is given at www.iana.org. Over 300 have been assigned. A few of the better known ones are listed in Fig 16-1.

Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial file transfer protocol
79	Finger	Lookup information about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

Figure 16-1. Some assigned ports.

16.3 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 65,516-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 1600 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. Although this protocol sounds simple, there are a number of sometimes subtle ins and outs, which we will cover below.

Segments can arrive out of order, so bytes 3072–4095 can arrive but cannot be acknowledged because bytes 2048–-3071 have not turned up yet.

Segments can also be delayed so long in transit that the sender times out and retransmits them.

The retransmissions may include different byte ranges than the original transmission, requiring a careful administration to keep track of which bytes have been correctly received so far. However, since each byte in the stream has its own unique offset, it can be done. TCP must be prepared to deal with these problems and solve them in an efficient way. A considerable amount of effort has gone into optimizing the performance of TCP streams, even in the face of network problems. A number of the algorithms used by many TCP implementations will be discussed below.

16.4 The TCP Segment Header

Fig 16-2 shows the layout of a TCP segment. 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.

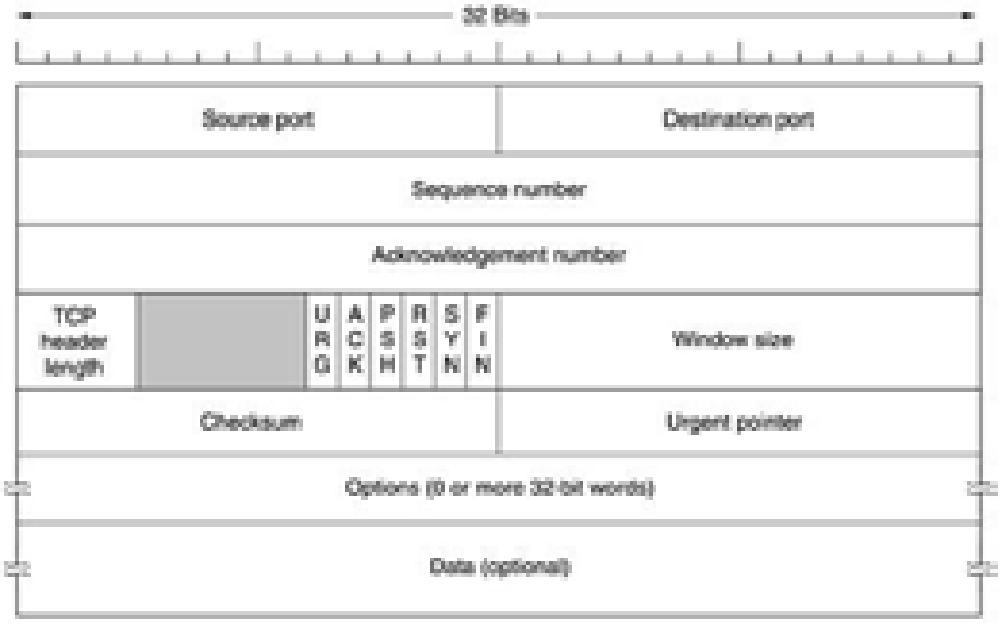


Figure 16-2. The TCP header.

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.

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. 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. 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.

The RST bit is used to reset a connection that has become confused due to a host crash or some other reason. 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.

Flow control in TCP is handled using a variable-sized sliding window. The Window size field tells how many bytes may be sent starting at the byte acknowledged. A Window size field of 0 is legal and says that the bytes up to and including Acknowledgement number - 1 have been received, but that the receiver is currently badly in need of a rest and would like no more data for the moment, thank you. The receiver can later grant permission to send by transmitting a segment with the same Acknowledgement number and a nonzero Window size field.

In most of the protocols, acknowledgements of frames received and permission to send new frames were tied together. This was a consequence of a fixed window size for each protocol. In TCP, acknowledgements and permission to send additional data are completely decoupled. In effect, a receiver can say: I have received bytes up through k but I do not want any more just now. This decoupling (in fact, a variable-sized window) gives additional flexibility.

A Checksum is also provided for extra reliability. It checksums the header, the data, and the conceptual pseudoheader as shown in Fig 16-3 When performing this computation, the TCP Checksum field is set to zero and the data field is padded out with an additional zero byte if its length is an odd number.

The checksum algorithm is simply to add up all the 16-bit words in one's complement and then to take the one's complement of the sum. As a consequence, when the receiver performs the calculation on the entire segment, including the Checksum field, the result should be 0.

Figure 16-3. The pseudo header included in the TCP checksum.

The pseudo header contains the 32-bit IP addresses of the source and destination machines, the protocol number for TCP (6), and the byte count for the TCP segment (including the header).

Including the pseudo header in the TCP checksum computation helps detect miss-delivered packets, but including it also violates the protocol hierarchy since the IP addresses in it belong to the IP layer, not to the TCP layer. UDP uses the same pseudoheader for its checksum.

The Options field provides a way to add extra facilities not covered by the regular header. The most important option is the one that allows each host to specify the maximum TCP payload it is willing to accept. Using large segments is more efficient than using small ones because the 20-byte header can then be amortized over more data, but small hosts may not be able to handle big segments.

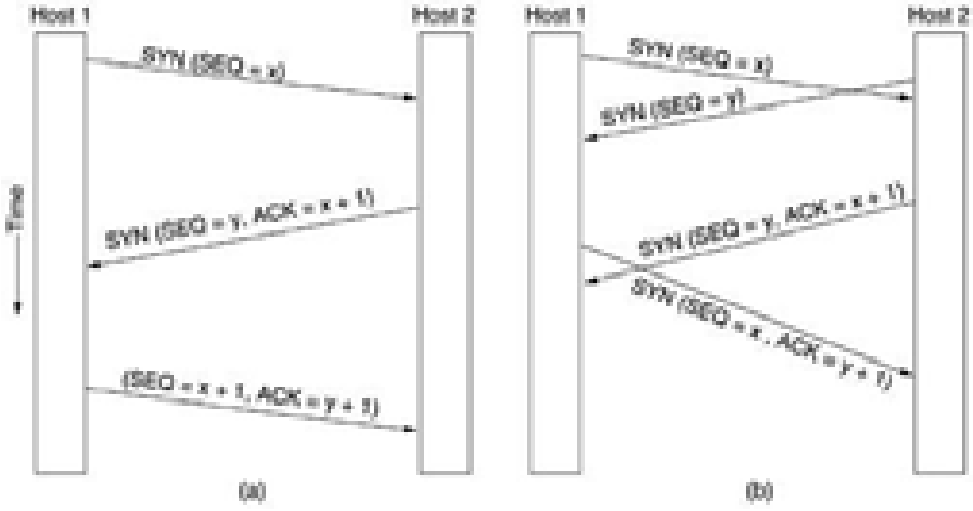
During connection setup, each side can announce its maximum and see its partner's. If a host does not use this option, it defaults to a 536-byte payload. All Internet hosts are required to accept TCP segments of 536 + 20 = 556 bytes. The maximum segment size in the two directions need not be the same.

16.5 TCP Connection Establishment

Connections are established in TCP by means of the three-way.

- 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.
- 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 sequence of TCP segments sent in the normal case is shown in Fig 16-4(a). Note that a SYN segment consumes 1 byte of sequence space so that it can be acknowledged unambiguously.



Figure

Fig 16-4**. (a) TCP connection establishment in the normal case. (b) Call collision.**

In the event that two hosts simultaneously attempt to establish a connection between the same two sockets, the sequence of events is as illustrated in Fig 16-4(b). 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.

16.6 TCP Connection Release

Although TCP connections are full duplex, to understand how connections are released it is best to think of them as a pair of simplex connections. Each simplex connection is released independently of its sibling.

To release a connection, either party can send a TCP segment with the FIN bit set, which means that it has no more data to transmit. When the FIN is acknowledged, that direction is shut down for new data. Data may continue to flow indefinitely in the other direction, however. When both directions have been shut down, the connection is released.

Normally, four TCP segments are needed to release a connection, one FIN and one ACK for each direction. However, it is possible for the first ACK and the second FIN to be contained in the same segment, reducing the total count to three. Just as with telephone calls in which both people say goodbye and hang

up the phone simultaneously, both ends of a TCP connection may send FIN segments at the same time.

These are each acknowledged in the usual way, and the connection is shut down. There is, in fact, no essential difference between the two hosts releasing sequentially or simultaneously. To avoid the two-army problem, timers are used. If a response to a FIN is not forthcoming within two maximum packet lifetimes, the sender of the FIN releases the connection.

The other side will eventually notice that nobody seems to be listening to it any more and will time out as well. While this solution is not perfect, given the fact that a perfect solution is theoretically impossible, it will have to do. In practice, problems rarely arise.

16.7 TCP Connection Management Modeling

The steps required to establish and release connections can be represented in a finite state machine with the 11 states listed in Fig 16-5. In each state, certain events are legal. When a legal event happens, some action may be taken. If some other event happens, an error is reported.

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

Figure 16-5. The states used in the TCP connection management finite state machine.

Each connection starts in the CLOSED state. It leaves that state when it does either a passive open (LISTEN), or an active open (CONNECT). If the other side does the opposite one, a connection is established and the state becomes ESTABLISHED. Connection release can be initiated by either side. When it is complete, the state returns to CLOSED.

The finite state machine itself is shown in Fig 16-6. The common case of a client actively connecting to a passive server is shown with heavy lines—solid for the client, dotted for the server. The lightface lines are unusual event sequences.

Each line in Fig 16-6 is marked by an event/action pair. The event can either be a user-initiated system call (CONNECT, LISTEN, SEND, or CLOSE), a segment arrival (SYN, FIN, ACK, or RST), or, in one case, a timeout of twice the

maximum packet lifetime. The action is the sending of a control segment (SYN, FIN, or RST) or nothing, indicated by —. Comments are shown in parentheses.

When an application program on the client machine issues a CONNECT request, the local TCP entity creates a connection record, marks it as being in the SYN SENT state, and sends a SYN segment.

Note that many connections may be open (or being opened) at the same time on behalf of multiple applications, so the state is per connection and recorded in the connection record. When the SYN+ACK arrives, TCP sends the final ACK of the three-way handshake and switches into the ESTABLISHED state. Data can now be sent and received.

When an application is finished, it executes a CLOSE primitive, which causes the local TCP entity to send a FIN segment and wait for the corresponding ACK (dashed box marked active close).

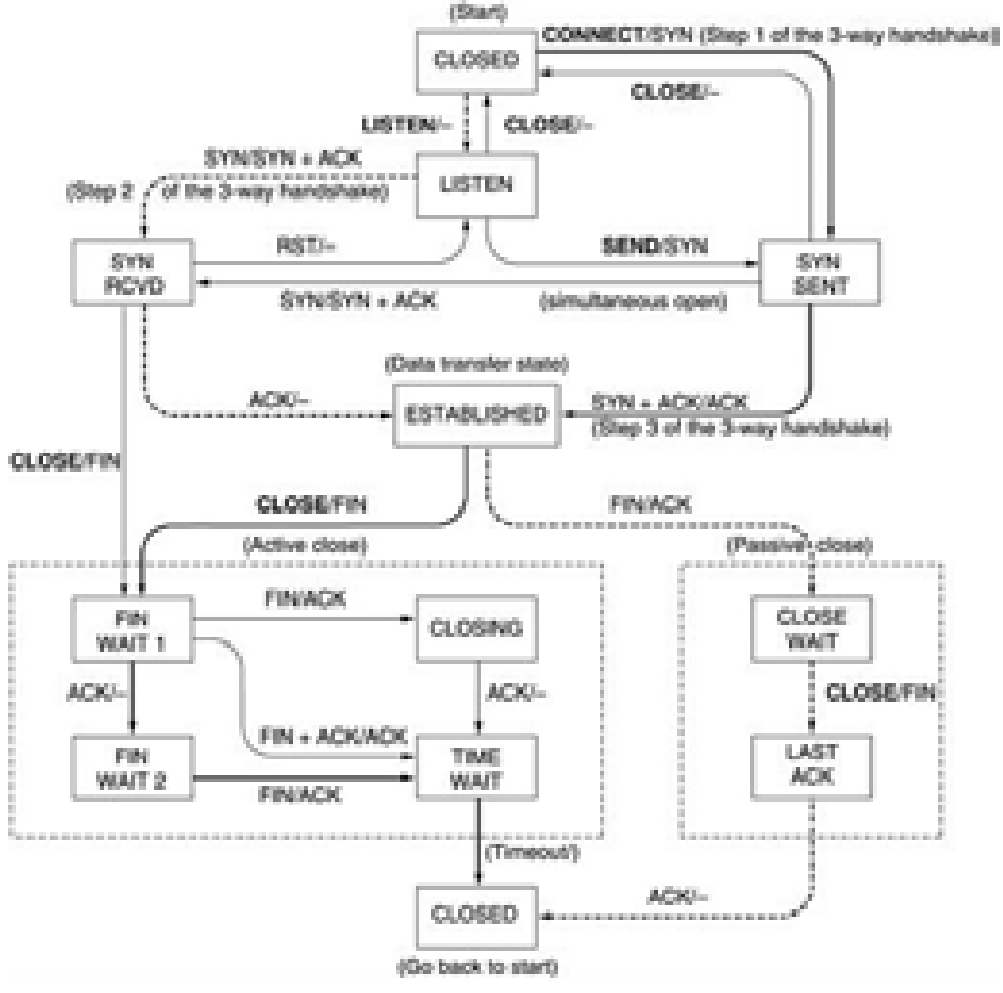


Figure 16-6. TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.

When the ACK arrives, a transition is made to state FIN WAIT 2 and one direction of the connection is now closed. When the other side closes, too, a FIN comes in, which is acknowledged. Now both sides are closed, but TCP waits a time equal to the maximum packet lifetime to guarantee that all packets from the connection have died off, just in case the acknowledgement was lost. When the timer goes off, TCP deletes the connection record.

Now let us examine connection management from the server's viewpoint.

- The server does a LISTEN and settles down to see who turns up.
- When a SYN comes in, it is acknowledged and the server goes to the SYN RCVD state.
- When the server's SYN is itself acknowledged, the three-way handshake is complete and the server goes to the ESTABLISHED state. Data transfer can now occur.
- When the client is done, it does a CLOSE, which causes a FIN to arrive at the server (dashed box marked passive close).
- The server is then signaled. When it, too, does a CLOSE, a FIN is sent to the client.
- When the client's acknowledgement shows up, the server releases the connection and deletes the connection record.

16.8 Summary

1. TCP is reliable and connection oriented network protocol.
2. The data over the TCP protocol is attached with TCP header information for the reliable communication.
3. The TCP connection establishment is done by the three way handshake method.
4. Connection release is performed by the four way acknowledgement packets.

UNIT – V

Lesson 17

DNS (The Domain Name System)

Contents

17.0 Aim

All equipments in the computer network have a unique ID associated with them. However, it is important that these equipments have names so that it is easy to identify them. Domain Name Server does the job of mapping the names of those equipments with their names. The objective of this chapter is to explain the details regarding Domain Name Servers and the concepts associated with it.

17.1 Introduction

Although programs theoretically could refer to hosts, mailboxes, and other resources by their network (e.g., IP) addresses, these addresses are hard for people to remember. Also, sending e-mail to tana@128.111.24.41 means that if Tana's ISP or organization moves the mail server to a different machine with a different IP address, her e-mail address has to change.

Consequently, ASCII names were introduced to decouple machine names from machine addresses. In this way, Tana's address might be something like tana@art.ucsb.edu. Nevertheless, the network itself understands only numerical addresses, so some mechanism is required to convert the ASCII strings to network addresses. In the following sections we will study how this mapping is accomplished in the Internet.

Way back in the ARPANET, there was simply a file, hosts.txt, that listed all the hosts and their IP addresses. Every night, all the hosts would fetch it from the site at which it was maintained. For a network of a few hundred large timesharing machines, this approach worked reasonably well. 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."

17.2 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 Mr. Kishore on Gandhi Nagar in Trichy, Tamilnadu and Mr. Kishore on Gandhi Nagar in Ahmadabad, Delhi. 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. All these domains can be represented by a tree, as shown in Fig. 17-1. 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.

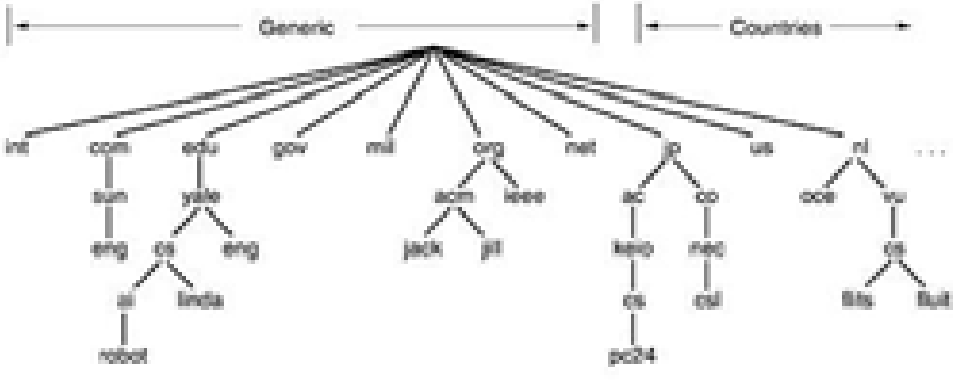


Figure 17-1. A portion of the Internet domain name space

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.

In November 2000, ICANN approved four new, general-purpose, top-level domains, namely, biz (businesses), info (information), name (people's names), and pro (professions, such as doctors and lawyers). In addition, three more specialized top-level domains were introduced at the request of certain industries. These are aero (aerospace industry), coop (co-operatives), and museum (museums). Other top-level domains will be added in the future.

In general, getting a second-level domain, such as name-of-company.com, is easy. It merely requires going to a registrar for the corresponding top-level domain (com in this case) to check if the desired name is available and not somebody else's trademark. If there are no problems, the requester pays a small annual fee and gets the name.

By now, virtually every common (English) word has been taken in the com domain. 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 eng.sun.com., rather than a UNIX-style name such as /com/sun/eng.

Notice that this hierarchical naming means that eng.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. In principle, domains can be inserted into the tree in two different ways. For example, cs.yale.edu could equally well be listed under the us country domain as cs.yale.ct.us. In practice, however, most organizations in the United States are under a generic domain, and most outside the United States are under the domain of their country. There is no rule against registering under two top-level domains, but few organizations except multinationals do it (e.g., sony.com and sony.nl).

Each domain controls how it allocates the domains under it. For example, Japan has domains ac.jp and co.jp that mirror edu and 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:

1. cs.yale.edu (Yale University, in the United States)
2. cs.vu.nl (Vrije Universiteit, in The Netherlands)
3. 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.

Naming follows organizational boundaries, not physical networks. For example, if the computer science and electrical engineering departments are located in the same building and share the same LAN, they can nevertheless have distinct domains. Similarly, even if computer science is split over Babbage Hall and Turing Hall, the hosts in both buildings will normally belong to the same domain.

17.3 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. 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. The Type field tells what kind of record this is. The most important types are listed in Fig. 17-2.

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.

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

Figure 17-2. . The principal DNS resource record types for IPv4.

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. The next most important record type is the MX record. It specifies the name of the host prepared to accept e-mail for the specified domain.

It is used because not every machine is prepared to accept e-mail. If someone wants to send e-mail to, for example, bill@microsoft.com, the sending host needs to find a mail server at microsoft.com that is willing to accept e-mail. The MX record can provide this information.

The NS records specify name servers. For example, every DNS database normally has an NS record for each of the top-level domains, so, for example, e-mail can be sent to distant parts of the naming tree.

CNAME records allow aliases to be created. For example, a person familiar with Internet naming in general and wanting to send a message to someone whose login name is paul in the computer science department at M.I.T. might guess that paul@cs.mit.edu will work.

Actually, this address will not work, because the domain for M.I.T.'s computer science department is lcs.mit.edu. However, as a service to people who do not know this, M.I.T. could create a CNAME entry to point people and programs in the right direction. An entry like this one might do the job:

```
cs.mit.edu  86400  IN  CNAME  lcs.mit.edu
```

Like CNAME, PTR points to another name. However, unlike CNAME, which is really just a macro definition, PTR is a regular DNS datatype whose interpretation depends on the context in which it is found. In practice, it is nearly always used to associate a name with an IP address to allow lookups of the IP address and return the name of the corresponding machine. These are called reverse lookups.

HINFO records allow people to find out what kind of machine and operating system a domain corresponds to. Finally, TXT records allow domains

to identify themselves in arbitrary ways. Both of these record types are for user convenience. Neither is required, so programs cannot count on getting them (and probably cannot deal with them if they do get them).

Finally, we have the Value field. This field can be a number, a domain name, or an ASCII string. The semantics depend on the record type. A short description of the Value fields for each of the principal record types is given in Fig. 17-2. For an example of the kind of information one might find in the DNS database of a domain, see Fig. 17-3.

This figure depicts part of a (semihypothetical) database for the cs.vu.nl domain shown in Fig. 17-1. The database contains seven types of resource records. The first noncomment line of Fig. 17-3 gives some basic information about the domain, which will not concern us further.

The next two lines give textual information about where the domain is located. Then come two entries giving the first and second places to try to deliver e-mail sent to person@cs.vu.nl. The zephyr (a specific machine) should be tried first. If that fails, the top should be tried as the next choice.

After the blank line, added for readability, come lines telling that the flits is a Sun workstation running UNIX and giving both of its IP addresses. Then three choices are given for handling e-mail sent to flits.cs.vu.nl. First choice is naturally the flits itself, but if it is down, the zephyr and top are the second and third choices.

Next comes an alias, www.cs.vu.nl, so that this address can be used without designating a specific machine. Creating this alias allows cs.vu.nl to change its World Wide Web server without invalidating the address people use to get to it. A similar argument holds for ftp.cs.vu.nl.

The next four lines contain a typical entry for a workstation, in this case, rowboat.cs.vu.nl. The information provided contains the IP address, the primary and secondary mail drops, and information about the machine. Then comes an entry for a non-UNIX system that is not capable of receiving mail itself, followed by an entry for a laser printer that is connected to the Internet.

What are not shown (and are not in this file) are the IP addresses used to look up the top-level domains. These are needed to look up distant hosts, but since they are not part of the cs.vu.nl domain, they are not in this file. They are supplied by the root servers, whose IP addresses are present in a system configuration file and loaded into the DNS cache when the DNS server is booted.

There are about a dozen root servers spread around the world, and each one knows the IP addresses of all the top-level domain servers. Thus, if a machine knows the IP address of at least one root server, it can look up any DNS name.

```
; Authoritative data for cs.vu.nl
cs.vu.nl.        86400  IN  SOA     star boss (9527,7200,7200,241920,86400)
cs.vu.nl.        86400  IN  TXT     "Divisie Wiskunde en Informatica."
cs.vu.nl.        86400  IN  TXT     "Vrije Universiteit Amsterdam."
cs.vu.nl.        86400  IN  MX      1 zephyr.cs.vu.nl.
cs.vu.nl.        86400  IN  MX      2 top.cs.vu.nl.

flits.cs.vu.nl.  86400  IN  HINFO   Sun Unix
flits.cs.vu.nl.  86400  IN  A       130.37.16.112
flits.cs.vu.nl.  86400  IN  A       192.31.231.165
flits.cs.vu.nl.  86400  IN  MX      1 flits.cs.vu.nl.
flits.cs.vu.nl.  86400  IN  MX      2 zephyr.cs.vu.nl.
flits.cs.vu.nl.  86400  IN  MX      3 top.cs.vu.nl.
www.cs.vu.nl.    86400  IN  CNAME   star.cs.vu.nl
ftp.cs.vu.nl.    86400  IN  CNAME   zephyr.cs.vu.nl

rowboat                 IN  A       130.37.56.201
                        IN  MX      1 rowboat
                        IN  MX      2 zephyr
                        IN  HINFO   Sun Unix

little-sister           IN  A       130.37.62.23
                        IN  HINFO   Mac MacOS

laserjet                IN  A       192.31.231.216
                        IN  HINFO   "HP Laserjet IIISi" Proprietary
```

Figure 17-3. A portion of a possible DNS database for cs.vu.nl

17.4 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. One possible way to divide the name space of Fig. 17-1 is shown in Fig. 17-4. Each zone contains some part of the tree and also contains name servers holding the information about that zone.

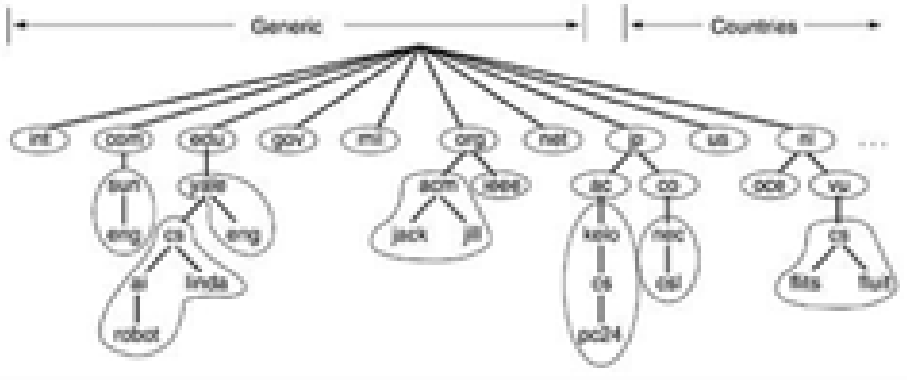


Figure 17-4. Part of the DNS name space showing the division into zones.

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. For example, in Fig. 17-4, 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 be made 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 is thus always correct. Authoritative records are in contrast to cached records, which may be out of date.

If, however, the domain is remote and no information about the requested domain is available locally, the name server sends a query message to the top-level name server for the domain requested. To make this process clearer, consider the example of Fig. 17-5. Here, a resolver on flits.cs.vu.nl wants to know the IP address of the host linda.cs.yale.edu.

In step 1, it sends a query to the local name server, cs.vu.nl. This query contains the domain name sought, the type (A) and the class (IN).

Figure 17-5. How a resolver looks up a remote name in eight steps

Let us suppose the local name server has never had a query for this domain before and knows nothing about it. It may ask a few other nearby name servers, but if none of them know, it sends a UDP packet to the server for edu given in its database (see Fig. 17-5), edu-server.net. It is unlikely that this server knows the address of linda.cs.yale.edu, and probably does not know cs.yale.edu either, but it must know all of its own children, so it forwards the request to the name server for yale.edu (step 3).

In turn, this one forwards the request to cs.yale.edu (step 4), which must have the authoritative resource records. Since each request is from a client to a server, the resource record requested works its way back in steps 5 through 8. Once these records get back to the cs.vu.nl name server, they will be entered into a cache there, in case they are needed later.

However, this information is not authoritative, since changes made at cs.yale.edu will not be propagated to all the caches in the world that may know

about it. For this reason, cache entries should not live too long. This is the reason that the Time_to_live field is included in each resource record. It tells remote name servers how long to cache records. If a certain machine has had the same IP address for years, it may be safe to cache that information for 1 day.

For more volatile information, it might be safer to purge the records after a few seconds or a minute. It is worth mentioning that the query method described here is known as a recursive query, since each server that does not have the requested information goes and finds it somewhere, then reports back.

An alternative form is also possible. In this form, when a query cannot be satisfied locally, the query fails, but the name of the next server along the line to try is returned. Some servers do not implement recursive queries and always return the name of the next server to try. It is also worth pointing out that when a DNS client fails to get a response before its timer goes off, it normally will try another server next time. The assumption here is that the server is probably down, rather than that the request or reply got lost.

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.

17.5 Summary

- Naming in the Internet uses a hierarchical scheme called the domain name system (DNS). At the top level are the well-known generic domains, including com and edu as well as about 200 country domains.
- DNS is implemented as a distributed database system with servers all over the world.
- DNS holds records with IP addresses, mail exchanges, and other information.
- By querying a DNS server, a process can map an Internet domain name onto the IP address used to communicate with that domain.

Lesson 18

Electronic Mail

Contents

18.0 Aim

To have an understanding about E-mail and the protocols used. E-Mail - Everyone in this world who uses computer will definitely know what it is. This makes people send mails from one end of this world to the other end within no span of time. This has definitely made people to stay closer wherever in the world they are. This chapter reveals what E-Mail is all about and some of the protocols that are used for exchanging the mails electronically.

18.1 Introduction

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 academic areas. 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.

Smiley	Meaning	Smiley	Meaning	Smiley	Meaning
:-)	I'm happy	=I:-)	Abe Lincoln	:+)	Big nose
:-(	I'm sad/angry	=]:-)	Uncle Sam	:-))	Double chin
:-|	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

Figure 18-1. Some smileys

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:

1. Sending a message to a group of people was inconvenient. Managers often need this facility to send memos to all their subordinates.
2. 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.
3. The originator (sender) never knew if a message arrived or not.
4. 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.
5. 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.
6. It was not possible to create and send messages containing a mixture of text, drawings, facsimile, and voice.

As experience was gained, more elaborate e-mail systems were proposed.

- ✓ In 1982, the ARPANET e-mail proposals were published as RFC 821 (transmission protocol) and RFC 822 (message format).
- ✓ Minor revisions, RFC 2821 and RFC 2822, have become Internet standards, but everyone still refers to Internet e-mail as RFC 822.
- ✓ In 1984, CCITT drafted its X.400 recommendation. After two decades of competition, e-mail systems based on RFC 822 are widely used, whereas those based on X.400 have disappeared.

The reason for RFC 822's success is not that it is so good, but that X.400 was so poorly designed and so complex that nobody could implement it well. Given a choice between a simple-minded, but working, RFC 822-based e-mail system and a supposedly truly wonderful, but nonworking, X.400 e-mail system, most organizations chose the former. Perhaps there is a lesson lurking in there somewhere. Consequently, our discussion of e-mail will focus on the Internet e-mail system.

18.2 Architecture and Services

In this section we will provide an overview of what e-mail systems can do and how they are organized. 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

- ✓ 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

- ✓ 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

- ✓ 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.

Displaying

- ✓ 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

- ✓ 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.

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.

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. Envelopes and messages are illustrated in Fig. 18-2.

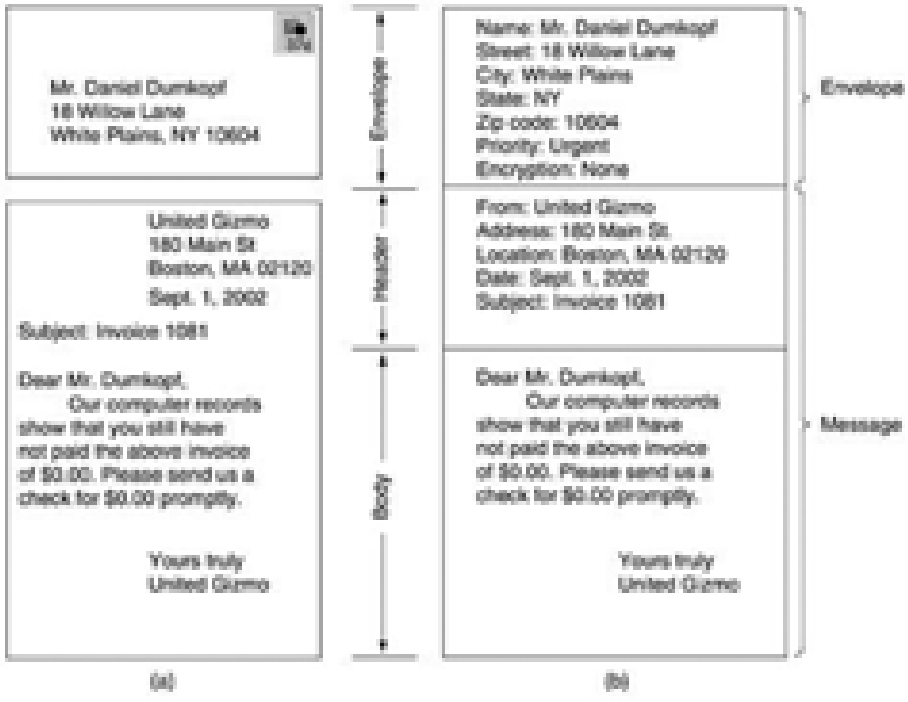


Figure 18-2 Envelopes and messages (a) Paper mail. (b) Electronic mail.

18.3 The User Agent

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.

18.3.1 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. 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.

18.3.2 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. 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. A display like that of Fig. 18-3 then appears on the screen. 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.

#	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

Figure 18-3. An example display of the contents of a mailbox

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.

18.4 Message Formats

18.4.1 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 in Fig. 18-4.

The To: 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.

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

Figure 18-4. RFC 822 header fields related to message transport.

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.

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

Figure 18-5. Some fields used in the RFC 822 message header.

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 as such and typically just contains the sender's address.

In addition to the fields of Fig. 18-4, 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. 18-5. 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.

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).

18.4.2 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

1. Messages in languages with accents (e.g., French and German).
2. Messages in non-Latin alphabets (e.g., Hebrew and Russian).
3. Messages in languages without alphabets (e.g., Chinese and Japanese).

4. 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.

- 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, as shown in Fig. 18-6.

- ✓ 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.
- ✓ The Content-Id: header identifies the content. It uses the same format as the standard Message-Id: header.

Header	Meaning
MIME-Version:	Identifies the MIME version
Content-Description:	Human-readable string telling what is in the message
Content-Id:	Unique identifier
Content-Transfer-Encoding:	How the body is wrapped for transmission
Content-Type:	Type and format of the content

Figure 18-6. RFC 822 headers added by MIME.

- ✓ The Content-Description: header is an ASCII string telling what is in the message. This header is needed so the recipient will know whether it is worth decoding and reading the message. If the string says: ''Photo of Barbara's hamster'' and the person getting the message is not a big hamster fan, the message will probably be discarded rather than decoded into a high-resolution color photograph.
- ✓ The Content-Transfer-Encoding: tells how the body is wrapped for transmission through a network that may object to most characters other than letters, numbers, and punctuation marks. Five schemes (plus an escape to new schemes) are provided. The simplest scheme is just ASCII text. ASCII characters use 7 bits and can be carried directly by the e-mail protocol provided that no line exceeds 1000 characters.

The next simplest scheme is the same thing, but using 8-bit characters, that is, all values from 0 up to and including 255. This encoding scheme violates the (original) Internet e-mail protocol but is used by some parts of the Internet that implement some extensions to the original protocol.

While declaring the encoding does not make it legal, having it explicit may at least explain things when something goes wrong. Messages using the 8-bit encoding must still adhere to the standard maximum line length.

Even worse are messages that use binary encoding. These are arbitrary binary files that not only use all 8 bits but also do not even respect the 1000-character line limit. Executable programs fall into this category. No guarantee is given that messages in binary will arrive correctly, but some people try anyway.

The correct way to encode binary messages is to use base64 encoding, sometimes called ASCII armor. In this scheme, groups of 24 bits are broken up into four 6-bit units, with each unit being sent as a legal ASCII character. The coding is "A" for 0, "B" for 1, and so on, followed by the 26 lower-case letters, the ten digits, and finally + and / for 62 and 63, respectively. The == and = sequences indicate that the last group contained only 8 or 16 bits, respectively.

Carriage returns and line feeds are ignored, so they can be inserted at will to keep the lines short enough. Arbitrary binary text can be sent safely using this scheme. For messages that are almost entirely ASCII but with a few non-ASCII characters, base64 encoding is somewhat inefficient. Instead, an encoding known as quoted-printable encoding is used. This is just 7-bit ASCII, with all the characters above 127 encoded as an equal sign followed by the character's value as two hexadecimal digits.

In summary, binary data should be sent encoded in base64 or quoted-printable form. When there are valid reasons not to use one of these schemes, it is possible to specify a user-defined encoding in the Content-Transfer-Encoding: header.

The last header shown in Fig. 18-6 is really the most interesting one. It specifies the nature of the message body. Seven types are defined in RFC 2045, each of which has one or more subtypes. The type and subtype are separated by a slash, as in

Content-Type: video/mpeg

The subtype must be given explicitly in the header; no defaults are provided. The initial list of types and subtypes specified in RFC 2045 is given in Fig. 18-7. Many new ones have been added since then, and additional entries are being added all the time as the need arises.

Header	Meaning
MIME-Version:	Identifies the MIME version
Content-Description:	Human-readable string telling what is in the message
Content-Id:	Unique identifier
Content-Transfer-Encoding:	How the body is wrapped for transmission
Content-Type:	Type and format of the content

Figure 18-7. The MIME types and subtypes defined in RFC 2045.

Let us now go briefly through the list of types. The text type is for straight ASCII text.

Text/Plain

The text/plain combination is for ordinary messages that can be displayed as received, with no encoding and no further processing.

Text/Enriched

The text/enriched subtype allows a simple markup language to be included in the text.

Text/XML

A subtype for the extensible markup language, text/xml, is defined in RFC 3023.

Image

The next MIME type is image, which is used to transmit still pictures.

PostScript

The other defined subtype is postscript, which refers to the PostScript language defined by Adobe Systems and widely used for describing printed pages.

Partial

The partial subtype makes it possible to break an encapsulated message into pieces and send them separately (for example, if the encapsulated message is too long). .

External-Body

The external-body subtype can be used for very long messages (e.g., video films).

Multipart

The final type is multipart, which allows a message to contain more than one part, with the beginning and end of each part being clearly delimited.

Mixed

The mixed subtype allows each part to be different, with no additional structure imposed.

Alternative

In contrast to multipart, the alternative subtype, allows the same message to be included multiple times but expressed in two or more different media.

A multimedia example is shown in Fig. 18-8. Here a birthday greeting is transmitted both as text and as a song. If the receiver has an audio capability, the user agent there will fetch the sound file, birthday.snd, and play it. If not, the lyrics are displayed on the screen in stony silence. The parts are delimited by two hyphens followed by a (software-generated) string specified in the boundary parameter.

```
From: elinor@abcd.com
To: carolyn@xyz.com
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@abcd.com>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Earth orbits sun integral number of times

This is the preamble. The user agent ignores it. Have a nice day.

--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/enriched

Happy birthday to you
Happy birthday to you
Happy birthday dear <bold> Carolyn </bold>
Happy birthday to you

--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
        access-type="anon-ftp";
        site="bicycle.abcd.com";
        directory="pub";
        name="birthday.snd"

content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--
```

Figure 18-8. A multipart message containing enriched and audio alternatives

Note that the Content-Type header occurs in three positions within this example. At the top level, it indicates that the message has multiple parts. Within each part, it gives the type and subtype of that part.

Finally, within the body of the second part, it is required to tell the user agent what kind of an external file it is to fetch. To indicate this slight difference in usage, we have used lower case letters here, although all headers are case insensitive. The content-transfer-encoding is similarly required for any external body that is not encoded as 7-bit ASCII.

Getting back to the subtypes for multipart messages, two more possibilities exist. The parallel subtype is used when all parts must be "viewed" simultaneously. For example, movies often have an audio channel and a video channel. Movies are more effective if these two channels are played back in parallel, instead of consecutively.

Finally, the digest subtype is used when many messages are packed together into a composite message. For example, some discussion groups on the Internet collect messages from subscribers and then send them out to the group as a single multipart/digest message.

18.5 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.

18.5.1 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. A sample dialog for sending the message of Fig. 18-8, including the numerical codes used by SMTP, is shown in Fig. 18-9. The lines sent by the client are marked C:. Those sent by the server are marked S:.

A few comments about Fig. 18-9 may be helpful. The first command from the client is indeed HELO. Of the various four-character abbreviations for HELLO, this one has numerous advantages over its biggest competitor. Why all the commands had to be four characters has been lost in the mists of time.

In Fig. 18-9, the message is sent to only one recipient, so only one RCPT command is used. Such commands are allowed to send a single message to multiple receivers. Each one is individually acknowledged or rejected. Even if some recipients are rejected (because they do not exist at the destination), the message can be sent to the other ones.

Finally, although the syntax of the four-character commands from the client is rigidly specified, the syntax of the replies is less rigid. Only the numerical code really counts. Each implementation can put whatever string it wants after the code. Even though the SMTP protocol is completely well defined, a few problems can still arise.

One problem relates to message length. Some older implementations cannot handle messages exceeding 64 KB. Another problem relates to timeouts. If the client and server have different timeouts, one of them may give up while the other is still busy, unexpectedly terminating the connection.

Finally, in rare situations, infinite mailstorms can be triggered. For example, if host 1 holds mailing list A and host 2 holds mailing list B and each list contains an entry for the other one, then a message sent to either list could generate a never-ending amount of e-mail traffic unless somebody checks for it.

To get around some of these problems, extended SMTP (ESMTP) has been defined in RFC 2821. Clients wanting to use it should send an EHLO message instead of HELO initially. If this is rejected, then the server is a regular SMTP server, and the client should proceed in the usual way. If the EHLO is accepted, then new commands and parameters are allowed.

Figure 18-9. Transferring a message from elinor@abcd.com to carolyn@xyz.com

18.6 Final Delivery

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.

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.

18.6.1 POP3

The situation that used to hold (both sender and receiver having a permanent connection to the Internet) is illustrated in Fig. 18-10(a). A situation in which the sender is (currently) on-line but the receiver is not is illustrated in Fig. 18-10(b).

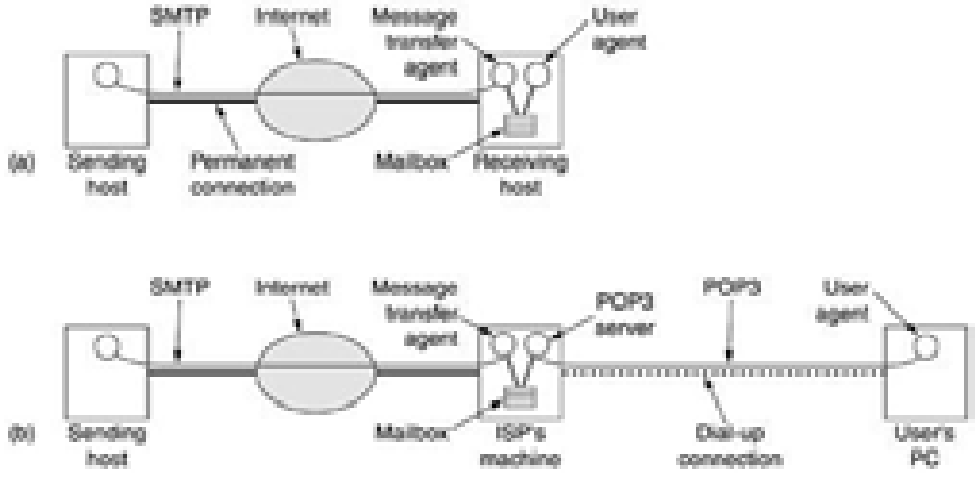


Figure 18-10. (a) Sending and reading mail when the receiver has a permanent Internet connection and the user agent runs on the same machine as the message transfer agent. (b) Reading e-mail when the receiver has a dial-up connection to an ISP.

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:

1. Authorization.
2. Transactions.
3. 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.

This behavior can be observed by typing something like:

```
telnet mail.isp.com 110
```

where mail.isp.com represents the DNS name of your ISP's mail server.

Telnet establishes a TCP connection to port 110, on which the POP3 server listens. Upon accepting the TCP connection, the server sends an ASCII message announcing that it is present.

Usually, it begins with +OK followed by a comment. An example scenario is shown in Fig. 18-11 starting after the TCP connection has been established. As before, the lines marked C: are from the client (user) and those marked S: are from the server (message transfer agent on the ISP's machine).

During the authorization state, the client sends over its user name and then its password. After a successful login, the client can then send over the LIST command, which causes the server to list the contents of the mailbox, one message per line, giving the length of that message. The list is terminated by a period.

Then the client can retrieve messages using the RETR command and mark them for deletion with DELE. When all messages have been retrieved (and possibly marked for deletion), the client gives the QUIT command to terminate the transaction state and enter the update state. When the server has deleted all the messages, it sends a reply and breaks the TCP connection.

```
                    S: +OK POP3 server ready
C: USER carolyn
                    S: +OK
C: PASS vegetables
                    S: +OK login successful
C: LIST
                    S: 1 2505
                    S: 2 14302
                    S: 3 8122
                    S: .
C: RETR 1
                    S: (sends message 1)
C: DELE 1
C: RETR 2
                    S: (sends message 2)
C: DELE 2
C: RETR 3
                    S: (sends message 3)
C: DELE 3
C: QUIT
                    S: +OK POP3 server disconnecting
```

Figure 18-11. Using POP3 to fetch three messages

While it is true that the POP3 protocol supports the ability to download a specific message or set of messages and leave them on the server, most e-mail programs just download everything and empty the mailbox. This behavior means that in practice, the only copy is on the user's hard disk. If that crashes, all e-mail may be lost permanently.

18.6.2 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 users.

This disadvantage gave rise to an alternative final delivery protocol, IMAP (Internet Message Access Protocol), which is defined in RFC 2060. Unlike POP3, which basically assumes 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. IMAP has many features, such as the ability to address mail not by arrival number as is done in Fig. 18-3, but by using attributes (e.g., give me the first message from Bobbie).

The general style of the IMAP protocol is similar to that of POP3 as shown in Fig. 18-11, except that are there dozens of commands. The IMAP server listens to port 143. A comparison of POP3 and IMAP is given in Fig. 18-12. It should be noted, however, that not every ISP supports both protocols and not every e-mail program supports both protocols. Thus, when choosing an e-mail program, it is important to find out which protocol(s) it supports and make sure the ISP supports at least one of them.

Feature	POP3	IMAP
Where is protocol defined	RFC 1939	RFC 2060
TCP port used	110	143
Where is e-mail stored	User's PC	Server
Where is e-mail read	Off-line	On-line
Connect time required	Little	Much
Use of server resources	Minimal	Extensive
Multiple mailboxes	No	Yes
Who backs up mailboxes	User	ISP
Good for mobile users	No	Yes
User control over downloading	Little	Great
Partial message downloads	No	Yes
Are disk quotas a problem	No	Could be in time
Simple to implement	Yes	No
Widespread support	Yes	Growing

Figure 18-12. A comparison of POP3 and IMAP

18.6.3 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.Still another common feature of final delivery is the ability to install a vacation daemon. This is a program that examines each incoming message and sends the sender an insipid reply such as

Hi. I'm on vacation. I'll be back on the 24th of August. Have a nice summer.

Such replies can also specify how to handle urgent matters in the interim, other people to contact for specific problems, etc. Most vacation daemons keep track of whom they have sent canned replies to and refrain from sending the same person a second reply.

The good ones also check to see if the incoming message was sent to a mailing list, and if so, do not send a canned reply at all. (People who send messages to large mailing lists during the summer probably do not want to get hundreds of replies detailing everyone's vacation plans.)

18.7 Webmail

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.
- If the login is successful, the server finds the user's mailbox and builds a listing similar to that of Fig. 18-3, only formatted as a Web page in HTML.

- 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.

18.8 Summary

- E-mail is one of the two killer apps for the Internet. Everyone from small children to grandparents now use it.
- Most e-mail systems in the world use the mail system now defined in RFCs 2821 and 2822.
- Messages sent in this system use system ASCII headers to define message properties.
- Many kinds of content can be sent using MIME.
- Messages are sent using SMTP, which works by making a TCP connection from the source host to the destination host and directly delivering the e-mail over the TCP connection.

Lesson 19

Cryptography

Contents

19.0 Aim

To learn about cryptography and the principles used in it. Security is a broad topic and covers a multitude of sins. In its simplest form, it is concerned with making sure that nosy people cannot read, or worse yet, secretly modify messages intended for other recipients. This chapter introduces cryptography and some the concepts like Quantum Cryptography.

19.1 Introduction

Cryptography comes from the Greek words for "secret writing." It has a long and colorful history going back thousands of years.

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.

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. These conflicting requirements have given rise to the model of Fig. 19-1.

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 cipher text, is then transmitted, often by messenger or radio.

We assume that the enemy, or intruder, hears and accurately copies down the complete cipher text. However, unlike the intended recipient, he does not know what the decryption key is and so cannot decrypt the cipher text 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).

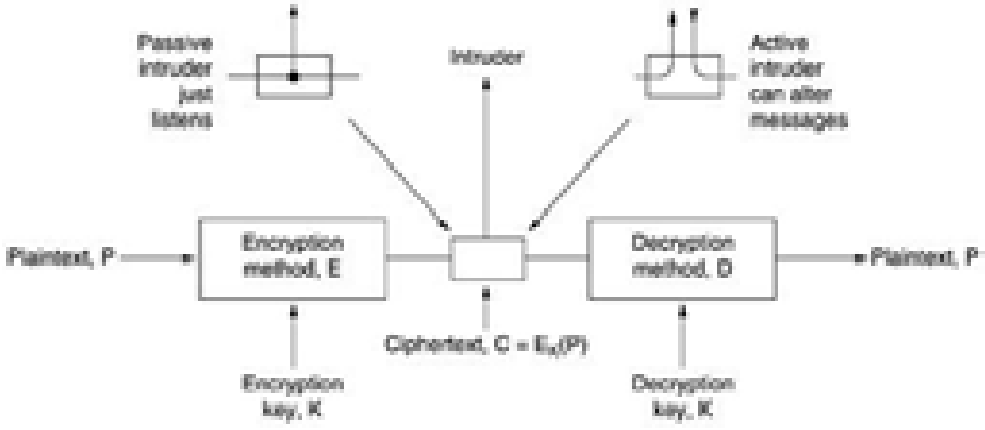


Figure 19-1. The encryption model (for a symmetric-key cipher)

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, cipher text, and keys. We will use $C = E_K(P)$ to mean that the encryption of the plaintext P using key K gives the cipher text C. Similarly, $P = D_K(C)$ represents the decryption of C to get the plaintext again. It then follows that

$$D_K(E_K(P)) = P$$

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, of Fig. 19-1 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.

This is where the key enters. The key consists of a (relatively) short string that selects one of many potential encryptions. In contrast to the general method, which may only be changed every few years, the key can be changed as often as required. Thus, our basic model is a stable and publicly-known general method parameterized by a secret and easily changed key. The idea that the cryptanalyst knows the algorithms and that the secrecy lies exclusively in the keys is called Kerckhoff's principle, named after the Flemish military cryptographer Auguste Kerckhoff who first stated it in 1883 (Kerckhoff, 1883). Thus, we have:

Kerckhoff's principle: All algorithms must be public; only the Keys are secret

The non-secrecy of the algorithm cannot be emphasized enough. Trying to keep the algorithm secret, known in the trade as security by obscurity, never works. Also, by publicizing the algorithm, the cryptographer gets free consulting from a large number of academic cryptologists eager to break the system so they can publish papers demonstrating how smart they are. If many experts have tried to break the algorithm for 5 years after its publication and no one has succeeded, it is probably pretty solid.

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 kind of 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?

Encryption methods have historically been divided into two categories: substitution ciphers and transposition ciphers. We will now deal with each of these briefly as background information for modern cryptography.

19.2 Difference between Ciphers and Codes

Professionals make a distinction between ciphers and codes.

Ciphers

A cipher is a character-for-character or bit-for-bit transformation, without regard to the linguistic structure of the message.

Codes

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 highly tonal,

exceedingly complex, and has no written form. And not a single person in Japan knew anything about it.

19.3 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, 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: Q 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! ≈ 4 x 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.

Nevertheless, given a surprisingly small amount of cipher text, the cipher can be broken easily. The basic attack takes advantage of the statistical properties of natural languages. In English, for example, e is the most common letter, followed by t, o, a, n, i, etc. The most common two-letter combinations, or digrams, are th, in, er, re, and an. The most common three-letter combinations, or trigrams, are the, ing, and, and ion.

A cryptanalyst trying to break a monoalphabetic cipher would start out by counting the relative frequencies of all letters in the ciphertext. Then he might tentatively assign the most common one to e and the next most common one to t. He would then look at trigrams to find a common one of the form tXe, which strongly suggests that X is h. Similarly, if the pattern thYt occurs frequently, the Y probably stands for a. With this information, he can look for a frequently occurring trigram of the form aZW, which is most likely and. By making guesses at common letters, digrams, and trigrams and knowing about likely patterns of vowels and consonants, the cryptanalyst builds up a tentative plaintext, letter by letter.

Another approach is to guess a probable word or phrase. For example, consider the following ciphertext from an accounting firm (blocked into groups of five characters):

CTBMN BYCTC BTJDS QXBNS GSTJC BTSWX CTQTZ CQVUJ
QJSGS TJQZZ MNQJS VLNSX VSZJU JDSTS JQUUS JUBXJ
DSKSU JSNTK BGAQJ ZBGYQ TLCTZ BNYBN QJSW

A likely word in a message from an accounting firm is financial. Using our knowledge that financial has a repeated letter (i), with four other letters between their occurrences, we look for repeated letters in the ciphertext at this spacing. We find 12 hits, at positions 6, 15, 27, 31, 42, 48, 56, 66, 70, 71, 76, and 82. However, only two of these, 31 and 42, have the next letter (corresponding to n in the plaintext) repeated in the proper place. Of these two, only 31 also has the correctly positioned, so we know that financial begins at position 30. From this point on, deducing the key is easy by using the frequency statistics for English text.

19.4 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. Figure 8-3 depicts a common transposition cipher, the columnar transposition. The cipher is keyed by a word or phrase not containing any repeated letters. In this example, MEGABUCK is the key.

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 cipher text is read out by columns, starting with the column whose key letter is the lowest.

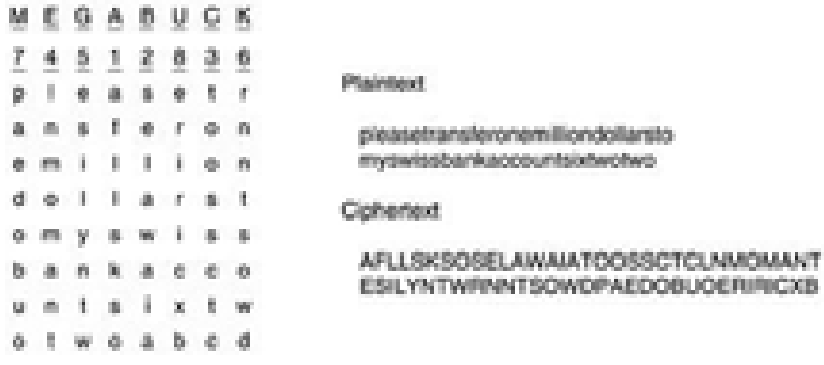


Figure 19-2. A transposition cipher

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 milliondollars occurs somewhere in the message. Observe that digrams MO, IL, LL, LA, IR and OS occur in the ciphertext as a result of this phrase wrapping around.

The ciphertext letter O follows the ciphertext 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 digrams MD, IO, LL, LL, IA, OR, and NS would have occurred instead.

In fact, for each key length, a different set of digrams is produced in the ciphertext. By hunting for the various possibilities, the cryptanalyst can often easily determine the key length. The remaining step is to order the columns. When the number of columns, k, is small, each of the k(k - 1) column pairs can be examined to see if its digram frequencies match those for English plaintext.

The pair with the best match is assumed to be correctly positioned. Now each remaining column is tentatively tried as the successor to this pair. The column whose digram and trigram frequencies give the best match is tentatively assumed to be correct.

The predecessor column is found in the same way. The entire process is continued until a potential ordering is found. Chances are that the plaintext will be recognizable at this point (e.g., if milloin occurs, it is clear what the error is). Some transposition ciphers accept a fixed-length block of input and produce a fixed-length block of output.

These ciphers can be completely described by giving a list telling the order in which the characters are to be output. For example, the cipher of Fig. 19-2 can be seen as a 64 character block cipher. Its output is 4, 12, 20, 28, 36, 44, 52, 60, 5, 13 , ... , 62. In other words, the fourth input character, a, is the first to be output, followed by the twelfth, f, and so on.

19.5 Symmetric-Key Algorithms

Modern cryptography uses the same basic ideas as traditional cryptography (transposition and substitution). 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.

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. Fig. 19-1 illustrates the use of a symmetric-key algorithm. 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 ciphertext.

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

circuits. Figure 19-3(a) shows a 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.

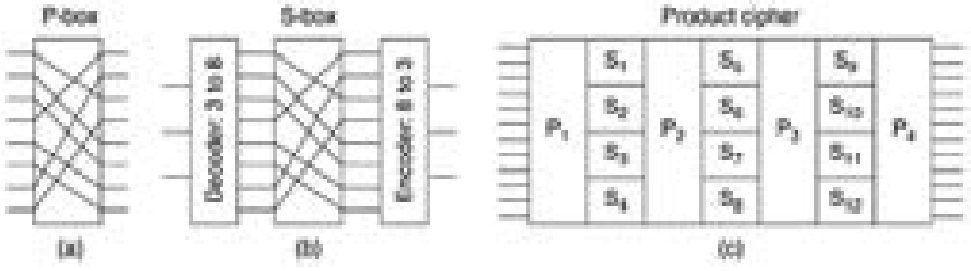


Figure 19-3. Basic elements of product ciphers. (a) P-box. (b) S-box. (c) Product.

Substitutions are performed by S-boxes, as shown in Fig. 19-3(b). In this example a 3-bit plaintext is entered and a 3-bit ciphertext is output. The 3-bit input selects one of the eight lines exiting from the first stage and sets it to 1; all the other lines are 0. The second stage is a P-box. The third stage encodes the selected input line in binary again. With the wiring shown, if the eight octal numbers 01234567 were input one after another, the output sequence would be 24506713. In other words, 0 has been replaced by 2, 1 has been replaced by 4, etc.

Again, by appropriate wiring of the P-box inside the S-box, any substitution can be accomplished. Furthermore, such a device can be built in hardware and can achieve great speed since encoders and decoders have only one or two (subnanosecond) gate delays and the propagation time across the P-box may well be less than 1 picosecond.

The real power of these basic elements only becomes apparent when we cascade a whole series of boxes to form a product cipher, as shown in Fig. 19-3(c). In this example, 12 input lines are transposed (i.e., permuted) by the first stage (P_1). Theoretically, it would be possible to have the second stage be an S-box that mapped a 12-bit number onto another 12-bit number. However, such a device would need $2^{12} = 4096$ crossed wires in its middle stage.

Instead, the input is broken up into four groups of 3 bits, each of which is substituted independently of the others. Although this method is less general, it is still powerful. By inclusion of a sufficiently large number of stages in the product cipher, the output can be made to be an exceedingly complicated function of the input.

Product ciphers that operate on k-bit inputs to produce k-bit outputs are very common. Typically, k is 64 to 256. A hardware implementation usually has at least 18 physical stages, instead of just seven as in Fig. 19-3(c). A software implementation is programmed as a loop with at least 8 iterations, each one

performing S-box-type substitutions on sub-blocks of the 64- to 256-bit data block, followed by a permutation that mixes the outputs of the S-boxes. Often there is a special initial permutation and one at the end as well. In the literature, the iterations are called rounds.

19.5.1 DES—The Data Encryption Standard

In January 1977, the U.S. Government adopted a product cipher developed by IBM as its official standard for unclassified information. 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.

An outline of DES is shown in Fig. 19-4(a). Plaintext is encrypted in blocks of 64 bits, yielding 64 bits of ciphertext. 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.

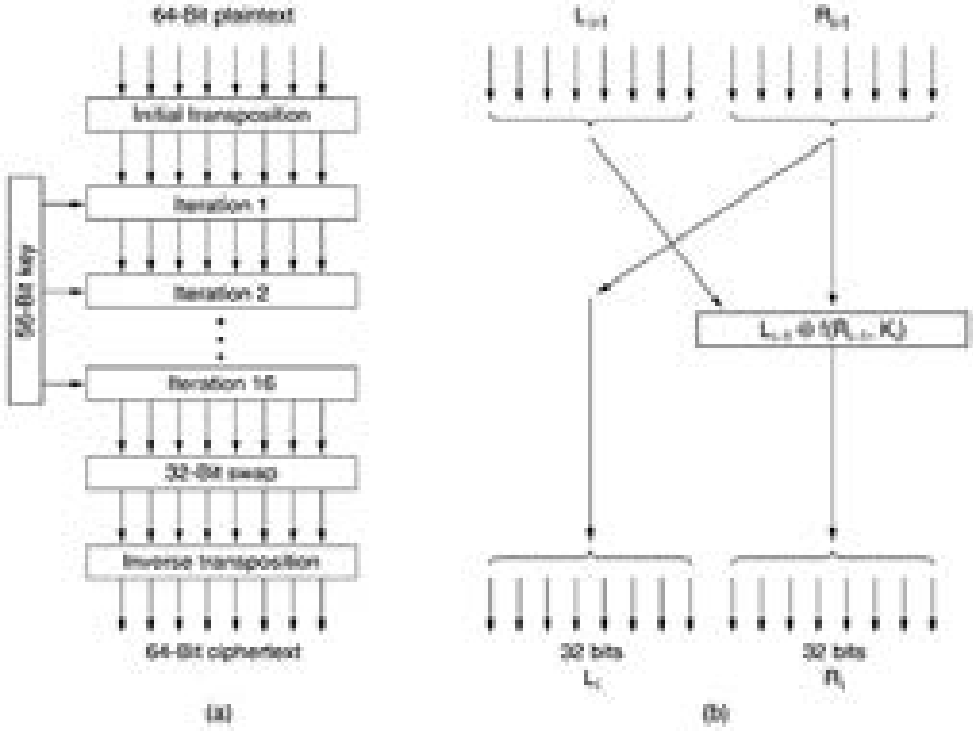


Figure 19-4. The data encryption standard. (a) General outline. (b) Detail of one iteration. The circled + means exclusive OR.

The operation of one of these intermediate stages is illustrated in Fig. 19-4(b). 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.

In each of the 16 iterations, a different key is used. Before the algorithm starts, a 56-bit transposition is applied to the key. Just before each iteration, the key is partitioned into two 28-bit units, each of which is rotated left by a number of bits dependent on the iteration number. K_i is derived from this rotated key by applying yet another 56-bit transposition to it. A different 48-bit subset of the 56 bits is extracted and permuted on each round.

A technique that is sometimes used to make DES stronger is called whitening. It consists of XORing a random 64-bit key with each plaintext block before feeding it into DES and then XORing a second 64-bit key with the resulting ciphertext before transmitting it. Whitening can easily be removed by running the reverse operations (if the receiver has the two whitening keys). Since this technique effectively adds more bits to the key length, it makes exhaustive search of the key space much more time consuming. Note that the same whitening key is used for each block (i.e., there is only one whitening key).

19.5.2 AES—The Advanced Encryption Standard

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 rules were:

1. The algorithm must be a symmetric block cipher.
2. The full design must be public.
3. Key lengths of 128, 192, and 256 bits must be supported.
4. Both software and hardware implementations must be possible.
5. 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:

1. Rijndael (from Joan Daemen and Vincent Rijmen, 86 votes).
2. Serpent (from Ross Anderson, Eli Biham, and Lars Knudsen, 59 votes).
3. Twofish (from a team headed by Bruce Schneier, 31 votes).
4. RC6 (from RSA Laboratories, 23 votes).
5. 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.

Rijndael

From a mathematical perspective, Rijndael is based on Galois field theory, which gives it some provable security properties. However, it can also be viewed as C code, without getting into the mathematics.

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. An outline of the code is given in Fig. 19-5.

```
#define LENGTH 16                               /* # bytes in data block or key */
#define NROWS 4                                 /* number of rows in state */
#define NCOLS 4                                 /* number of columns in state */
#define ROUNDS 10                               /* number of iterations */
typedef unsigned char byte;                     /* unsigned 8-bit integer */

rijndael(byte plaintext[LENGTH], byte ciphertext[LENGTH], byte key[LENGTH])
{
  int r;                                        /* loop index */
  byte state[NROWS][NCOLS];                     /* current state */
  struct {byte k[NROWS][NCOLS];} rk[ROUNDS + 1];     /* round keys */

  expand_key(key, rk);                          /* construct the round keys */
  copy_plaintext_to_state(state, plaintext);    /* init current state */
  xor_roundkey_into_state(state, rk[0]);        /* XOR key into state */

  for (r = 1; r <= ROUNDS; r++) {
      substitute(state);                        /* apply S-box to each byte */
      rotate_rows(state);                       /* rotate row i by i bytes */
      if (r < ROUNDS) mix_columns(state);       /* mix function */
      xor_roundkey_into_state(state, rk[r]);    /* XOR key into state */
  }
  copy_state_to_ciphertext(ciphertext, state);  /* return result */
}
```

Figure 19-5. An outline of Rijndael.

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).

The next step is to copy the plaintext into the state array so it can be processed during the rounds. It is copied in column order, with the first four bytes going into column 0, the next four bytes going into column 1, and so on. Both the columns and the rows are numbered starting at 0, although the rounds are numbered starting at 1. This initial setup of the 12 byte arrays of size 4 x 4 is illustrated in Fig. 19-6.

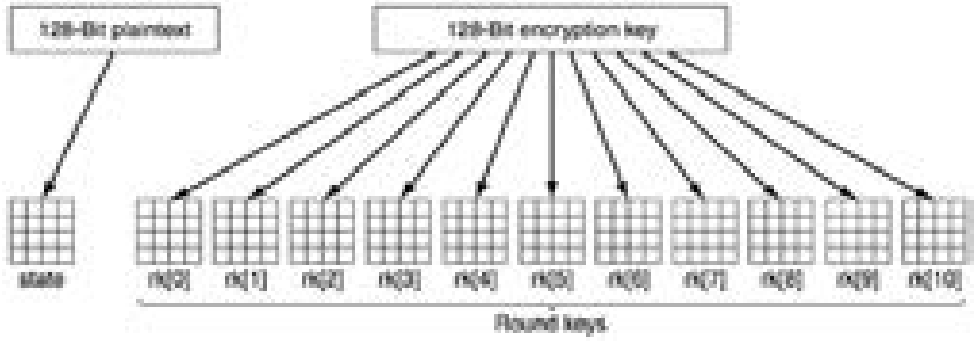


Figure 19-6. Creating of the state and rk arrays.

There is one more step before the main computation begins: rk[0] is XORed into state byte for byte. In other words each of the 16 bytes in state is replaced by the XOR of itself and the corresponding byte in rk[0].

Now it is time for the main attraction. The loop executes 10 iterations, one per round, transforming state on each iteration. The contents of each round consist of four steps. Step 1 does a byte-for-byte substitution on state. Each byte in turn is used as an index into an S-box to replace its value by the contents of that S-box entry. This step is a straight monoalphabetic substitution cipher. Unlike DES, which has multiple S-boxes, Rijndael has only one S-box.

Step 2 rotates each of the four rows to the left. Row 0 is rotated 0 bytes (i.e., not changed), row 1 is rotated 1 byte, row 2 is rotated 2 bytes, and row 3 is rotated 3 bytes. This step diffuses the contents of the current data around the block, analogous to the permutations of Fig. 19-3.

Step 3 mixes up each column independently of the other ones. The mixing is done using matrix multiplication in which the new column is the product of the old column and a constant matrix, with the multiplication done using the finite Galois field, GF(2^8). Although this may sound complicated, an algorithm exists that allows each element of the new column to be computed using two table lookups and three XORs (Daemen and Rijmen, 2002, Appendix E).

Finally, step 4 XORs the key for this round into the state array.

Since every step is reversible, decryption can be done just by running the algorithm backward. However, there is also a trick available in which decryption can be done by running the encryption algorithm, using different tables.

The algorithm has been designed not only for great security, but also for great speed. A good software implementation on a 2-GHz machine should be able to achieve an encryption rate of 700 Mbps, which is fast enough to encrypt over 100 MPEG-2 videos in real time. Hardware implementations are faster still.

19.6 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 digram, 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.

An example of how one-time pads are used is given in Fig. 19-7. 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 cipher text.

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.

Figure 19-7. The use of a one-time pad for encryption and the possibility of getting any possible plaintext from the cipher text by the use of some other pad

Disadvantages of One-time pad:

- 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.
- The total amount of data that can be transmitted is limited by the amount of key available. If the spy strikes it rich and discovers a wealth of data, he may find himself unable to transmit it back to headquarters because the key has been used up.
- The sensitivity of the method to lost or inserted characters. If the sender and receiver get out of synchronization, all data from then on will appear garbled.

With the advent of computers, the one-time pad might potentially become practical for some applications. The source of the key could be a special DVD that contains several gigabytes of information and if transported in a DVD movie box and prefixed by a few minutes of video, would not even be suspicious. Of course, at gigabit network speeds, having to insert a new DVD every 30 sec could become tedious. And the DVDs must be personally carried

from the sender to the receiver before any messages can be sent, which greatly reduces their practical utility.

19.6.1 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).

A user, Alice, wants to establish a one-time pad with a second user, Bob. Alice and Bob are called principals, the main characters in our story. For example, Bob is a banker with whom Alice would like to do business. The names ''Alice'' and ''Bob'' have been used for the principals in virtually every paper and book on cryptography in the past decade. Cryptographers love tradition. If we were to use ''Andy'' and ''Barbara'' as the principals, no one would believe anything in this chapter. So be it.

If Alice and Bob could establish a one-time pad, they could use it to communicate securely. The question is: How can they establish it without previously exchanging DVDs? We can assume that Alice and Bob are at opposite ends of an optical fiber over which they can send and receive light pulses. However, an intrepid intruder, Trudy, can cut the fiber to splice in an active tap. Trudy can read all the bits in both directions. She can also send false messages in both directions. The situation might seem hopeless for Alice and Bob, but quantum cryptography can shed some new light on the subject.

Quantum cryptography – Concept

Quantum cryptography is based on the fact that light comes in little packets called photons, which have some peculiar properties. Furthermore, light can be polarized by being passed through a polarizing filter, a fact well known to both sunglasses wearers and photographers.

If a beam of light (i.e., a stream of photons) is passed through a polarizing filter, all the photons emerging from it will be polarized in the direction of the filter's axis (e.g., vertical). If the beam is now passed through a second polarizing filter, the intensity of the light emerging from the second filter is proportional to the square of the cosine of the angle between the axes.

If the two axes are perpendicular, no photons get through. The absolute orientation of the two filters does not matter; only the angle between their axes counts. To generate a one-time pad, Alice needs two sets of polarizing filters. Set one consists of a vertical filter and a horizontal filter. This choice is called a rectilinear basis. A basis (plural: bases) is just a coordinate system.

The second set of filters is the same, except rotated 45 degrees, so one filter runs from the lower left to the upper right and the other filter runs from the upper left to the lower right. This choice is called a diagonal basis. Thus, Alice has two bases, which she can rapidly insert into her beam at will.

In reality, Alice does not have four separate filters, but a crystal whose polarization can be switched electrically to any of the four allowed directions at great speed. Bob has the same equipment as Alice. The fact that Alice and Bob each have two bases available is essential to quantum cryptography.

For each basis, Alice now assigns one direction as 0 and the other as 1. In the example presented below, we assume she chooses vertical to be 0 and horizontal to be 1. Independently, she also chooses lower left to upper right as 0 and upper left to lower right as 1. She sends these choices to Bob as plaintext.

Now Alice picks a one-time pad, for example based on a random number generator (a complex subject all by itself). She transfers it bit by bit to Bob, choosing one of her two bases at random for each bit.

To send a bit, her photon gun emits one photon polarized appropriately for the basis she is using for that bit. For example, she might choose bases of diagonal, rectilinear, rectilinear, diagonal, rectilinear, etc. To send her one-time pad of 1001110010100110 with these bases, she would send the photons shown in Fig. 19-8(a). Given the one-time pad and the sequence of bases, the polarization to use for each bit is uniquely determined. Bits sent one photon at a time are called qubits.

Bob does not know which bases to use, so he picks one at random for each arriving photon and just uses it, as shown in Fig. 19-8(b). If he picks the correct basis, he gets the correct bit. If he picks the incorrect basis, he gets a random bit because if a photon hits a filter polarized at 45 degrees to its own polarization, it randomly jumps to the polarization of the filter or to a polarization perpendicular to the filter with equal probability. This property of photons is fundamental to quantum mechanics.

Thus, some of the bits are correct and some are random, but Bob does not know which are which. Bob's results are depicted in Fig. 19-8(c). How does Bob find out which bases he got right and which he got wrong? He simply tells Alice which basis he used for each bit in plaintext and she tells him which are right and which are wrong in plaintext, as shown in Fig. 19-8(d).

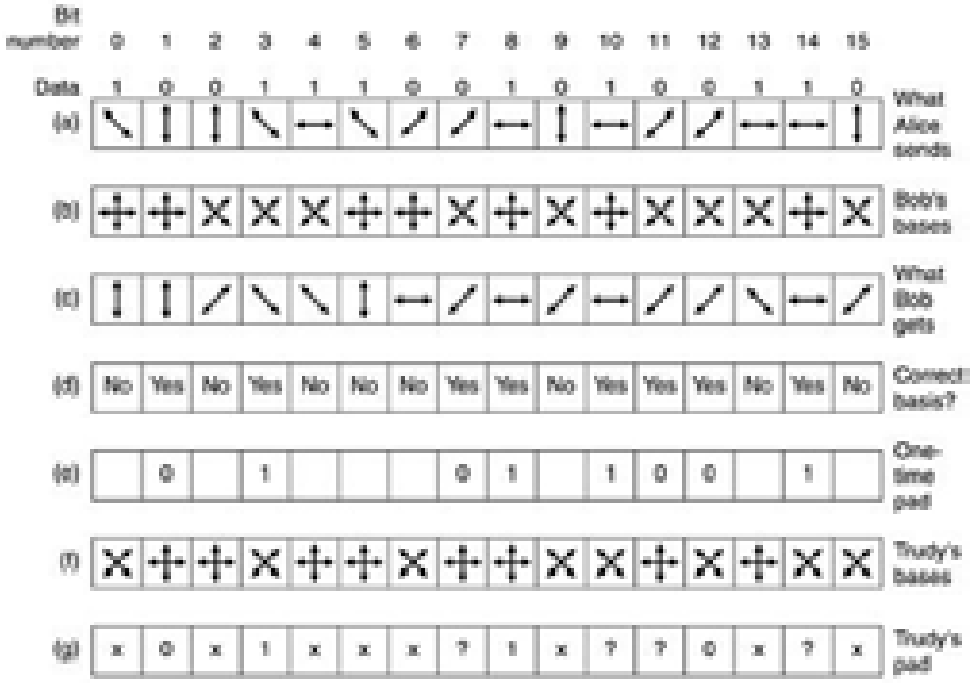


Figure 19-8. An example of quantum cryptography

From this information both of them can build a bit string from the correct guesses, as shown in Fig. 19-8(e). On the average, this bit string will be half the length of the original bit string, but since both parties know it, they can use it as a one-time pad. All Alice has to do is transmit a bit string slightly more than twice the desired length and she and Bob have a one-time pad of the desired length. Problem solved.

19.6.2 Overcoming the problem of intrusion

Consider the presence of Trudy. Suppose that she is curious about what Alice has to say and cuts the fiber, inserting her own detector and transmitter. Unfortunately for her, she does not know which basis to use for each photon either. The best she can do is pick one at random for each photon, just as Bob does.

An example of her choices is shown in Fig. 19-8(f). When Bob later reports (in plaintext) which bases he used and Alice tells him (in plaintext) which ones are correct, Trudy now knows when she got it right and when she got it wrong.

In Fig. 19-8 she got it right for bits 0, 1, 2, 3, 4, 6, 8, 12, and 13. But she knows from Alice's reply in Fig. 19-8(d) that only bits 1, 3, 7, 8, 10, 11, 12, and 14 are part of the one-time pad. For four of these bits (1, 3, 8, and 12), she guessed right and captured the correct bit. For the other four (7, 10, 11, and 14) she guessed wrong and does not know the bit transmitted. Thus, Bob

knows the one-time pad starts with 01011001, from Fig. 19-8(e) but all Trudy has is 01?1??0?, from Fig. 19-8(g).

Of course, Alice and Bob are aware that Trudy may have captured part of their one-time pad, so they would like to reduce the information Trudy has. They can do this by performing a transformation on it. For example, they could divide the one-time pad into blocks of 1024 bits and square each one to form a 2048-bit number and use the concatenation of these 2048-bit numbers as the one-time pad.

With her partial knowledge of the bit string transmitted, Trudy has no way to generate its square and so has nothing. The transformation from the original one-time pad to a different one that reduces Trudy's knowledge is called privacy amplification. In practice, complex transformations in which every output bit depends on every input bit are used instead of squaring.

Poor Trudy. Not only does she have no idea what the one-time pad is, but her presence is not a secret either. After all, she must relay each received bit to Bob to trick him into thinking he is talking to Alice.

The trouble is, the best she can do is transmit the qubit she received, using the polarization she used to receive it, and about half the time she will be wrong, causing many errors in Bob's one-time pad.

When Alice finally starts sending data, she encodes it using a heavy forward-error-correcting code. From Bob's point of view, a 1-bit error in the one-time pad is the same as a 1-bit transmission error. Either way, he gets the wrong bit. If there is enough forward error correction, he can recover the original message despite all the errors, but he can easily count how many errors were corrected.

If this number is far more than the expected error rate of the equipment, he knows that Trudy has tapped the line and can act accordingly (e.g., tell Alice to switch to a radio channel, call the police, etc.). If Trudy had a way to clone a photon so she had one photon to inspect and an identical photon to send to Bob, she could avoid detection, but at present no way to clone a photon perfectly is known.

But even if Trudy could clone photons, the value of quantum cryptography to establish one-time pads would not be reduced. Although quantum cryptography has been shown to operate over distances of 60 km of fiber, the equipment is complex and expensive. Still, the idea has promise.

19.7 Two Fundamental Cryptographic Principles

Redundancy

The first principle is that all encrypted messages must contain some redundancy, that is, information not needed to understand the message.

However, adding redundancy also makes it easier for cryptanalysts to break messages. Suppose that the mail order business is highly competitive, and The Couch Potato's main competitor, The Sofa Tuber, would dearly love to know how many sandboxes TCP is selling. Consequently, they have tapped TCP's telephone line. In the original scheme with 3-byte messages, cryptanalysis was nearly impossible, because after guessing a key, the

cryptanalyst had no way of telling whether the guess was right. After all, almost every message is technically legal. With the new 12-byte scheme, it is easy for the cryptanalyst to tell a valid message from an invalid one. Thus, we have

Cryptographic principle 1: Messages must contain some redundancy

In other words, upon decrypting a message, the recipient must be able to tell whether it is valid by simply inspecting it and perhaps performing a simple computation. This redundancy is needed to prevent active intruders from sending garbage and tricking the receiver into decrypting the garbage and acting on the "plaintext."

However, this same redundancy makes it much easier for passive intruders to break the system, so there is some tension here. Furthermore, the redundancy should never be in the form of n zeros at the start or end of a message, since running such messages through some cryptographic algorithms gives more predictable results, making the cryptanalysts' job easier. A CRC polynomial is much better than a run of 0s since the receiver can easily verify it, but it generates more work for the cryptanalyst. Even better is to use a cryptographic hash, a concept we will explore later.

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. 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.

19.8 Summary

- Cryptography is a tool that can be used to keep information confidential and to ensure its integrity and authenticity.
- All modern cryptographic systems are based on Kerckhoff's principle of having a publicly-known algorithm and a secret key.
- Many cryptographic algorithms use complex transformations involving substitutions and permutations to transform the plaintext into the ciphertext.
- However, if quantum cryptography can be made practical, the use of one-time pads may provide truly unbreakable cryptosystems.

Lesson 20

Public-Key Algorithms

Contents

20.0 Aim

As millions of ordinary citizens are using networks for banking, shopping, and filing their tax returns, network security is looming on the horizon as a potentially massive problem. In this chapter, we will discuss some of the algorithms that are used to implement security in computer networks.

20.1 Introduction

Historically, distributing the keys has always been the weakest link in most cryptosystems. No matter how strong a cryptosystem was, if an intruder could steal the key, the system was worthless. Cryptologists always took for granted that the encryption key and decryption key were the same (or easily derived from one another). But the key had to be distributed to all users of the system. Thus, it seemed as if there was an inherent built-in problem. Keys had to be protected from theft, but they also had to be distributed, so they could not just be locked up in a bank vault.

In 1976, two researchers at Stanford University, Diffie and Hellman (1976), proposed a radically new kind of cryptosystem, one in which the encryption and decryption keys were different, and the decryption key could not feasibly be derived from the encryption key. 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:

1. D(E(P)) = P.
2. It is exceedingly difficult to deduce D from E.
3. 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.

Now let us see if we can solve the problem of establishing a secure channel between Alice and Bob, who have never had any previous contact. Both Alice's encryption key, E_A, and Bob's encryption key, E_B, are assumed to be in publicly readable files.

Now Alice takes her first message, P, computes $E_B(P)$, and sends it to Bob. Bob then decrypts it by applying his secret key D_B [i.e., he computes $D_B(E_B(P)) = P$]. No one else can read the encrypted message, $E_B(P)$, because the encryption system is assumed strong and because it is too difficult to derive D_B from the publicly known E_B. To send a reply, R, Bob transmits $E_A(R)$. Alice and Bob can now communicate securely.

A note on terminology is perhaps useful here. Public-key cryptography requires each user to have two keys: a public key, used by the entire world for encrypting messages to be sent to that user, and a private key, which the user needs for decrypting messages. We will consistently refer to these keys as the public and private keys, respectively, and distinguish them from the secret keys used for conventional symmetric-key cryptography.

20.2 RSA

The important need is 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.

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.

1. Choose two large primes, p and q (typically 1024 bits).
2. Compute n = p x q and z = (p - 1) x (q - 1).
3. Choose a number relatively prime to z and call it d.
4. 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 $0 \le P < n$.
- ✓ 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 is given in Fig. 5-4-1.

- ✓ 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 (mod 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 $P = C^7 \pmod{33}$.
- ✓ The figure shows the encryption of the plaintext "SUZANNE" as an example.

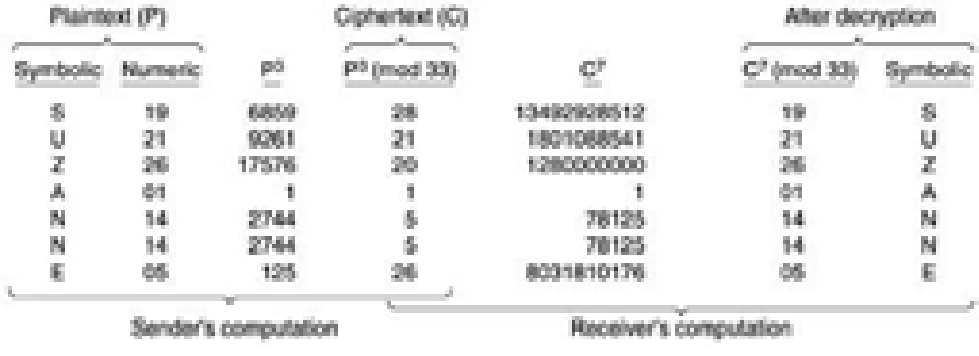


Figure 20-1. An example of the RSA algorithm

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 mono-alphabetic substitution cipher, not very impressive. If instead we had chosen p and q 2^{512}, we would have n 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.

It should be pointed out that using RSA as we have described is similar to using a symmetric algorithm in ECB mode—the same input block gives the same output block. Therefore, some form of chaining is needed for data encryption. However, in practice, most RSA-based systems use public-key cryptography primarily for distributing one-time session keys for use with some symmetric-key algorithm such as AES or triple DES. RSA is too slow for actually encrypting large volumes of data but is widely used for key distribution.

20.3 Other Public-Key Algorithms

Although RSA is widely used, it is by no means the only public-key algorithm known. Some of the other algorithms are discussed below.

20.3.1 Knapsack Algorithm

The first public-key algorithm was the knapsack algorithm (Merkle and Hellman, 1978). The idea here is that someone owns a large number of objects, each with a different weight.

The owner encodes the message by secretly selecting a subset of the objects and placing them in the knapsack. The total weight of the objects in the knapsack is made public, as is the list of all possible objects. The list of objects in the knapsack is kept secret.

With certain additional restrictions, the problem of figuring out a possible list of objects with the given weight was thought to be computationally infeasible and formed the basis of the public-key algorithm. The algorithm's inventor, Ralph Merkle, was quite sure that this algorithm could not be broken, so he offered a $100 reward to anyone who could break it.

Adi Shamir (the ''S'' in RSA) promptly broke it and collected the reward. Undeterred, Merkle strengthened the algorithm and offered a $1000 reward to anyone who could break the new one. Ronald Rivest (the ''R'' in RSA) promptly broke the new one and collected the reward. Merkle did not dare offer $10,000 for the next version, so ''A'' (Leonard Adleman) was out of luck. Nevertheless, the knapsack algorithm is not considered secure and is not used in practice any more.

Other public-key schemes are based on the difficulty of computing discrete logarithms. Algorithms that use this principle have been invented by El Gamal (1985) and Schnorr (1991).

A few other schemes exist, such as those based on elliptic curves (Menezes and Vanstone, 1993), but the two major categories are those based on the difficulty of factoring large numbers and computing discrete logarithms modulo a large prime. These problems are thought to be genuinely difficult to

solve—mathematicians have been working on them for many years without any great break-through.

20.4 Summary

Public-key algorithms have the property that different keys are used for encryption and decryption and that the decryption key cannot be derived from the encryption key. These properties make it possible to publish the public key. The main public-key algorithm is RSA, which derives its strength from the fact that it is very difficult to factor large numbers.

Lesson 21

Digital Signatures

Contents

21.0 Aim

Digital Signatures play an important role in order to verify whether the information received is from the correct and known source. Moreover, the users who send the data cannot deny that they have not sent it due to the fact that they have signed it digitally. This chapter gives an introduction about the Digital Signatures and their types.

21.1 Introduction

The authenticity of many legal, financial, and other documents is determined by the presence or absence of an authorized handwritten signature. 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 unforgettable 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:

1. The receiver can verify the claimed identity of the sender.
2. The sender cannot later repudiate the contents of the message.
3. 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 nonrepudiation. The digital signature schemes that we will now study help provide it.

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.

21.2 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 timestamp to ensure freshness, and $K_A(B, R_A, t, P)$ is the message encrypted with her key, K_A. Then she sends it as depicted in Fig. 21-1. BB sees that the message is from Alice, decrypts it, and sends a message to Bob as shown. The message to Bob contains the plaintext of Alice's message and also the signed message K_{BB} (A, t, P). Bob now carries out Alice's request.

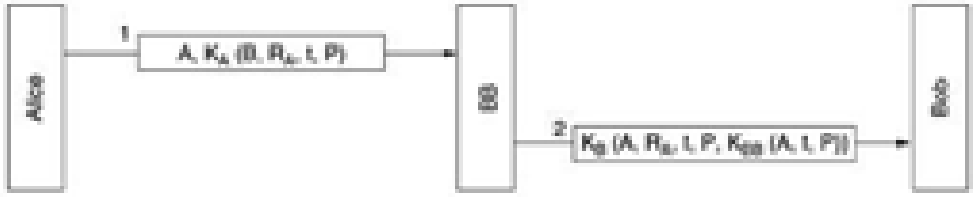


Figure 21-1. Digital signatures with Big Brother

What happens if Alice later denies sending the message? Step 1 is that everyone sues everyone. 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.

One potential problem with the signature protocol of Fig. 21-1 is Trudy replaying either message. To minimize this problem, timestamps are used throughout. Furthermore, Bob can check all recent messages to see if R_A was used in any of them. If so, the message is discarded as a replay. Note that based on the timestamp, Bob will reject very old messages. To guard against instant replay attacks, Bob just checks the R_A of every incoming message to see

if such a message has been received from Alice in the past hour. If not, Bob can safely assume this is a new request.

21.3 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. 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 $D_A(P)$, as shown in Fig. 21-2. 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.

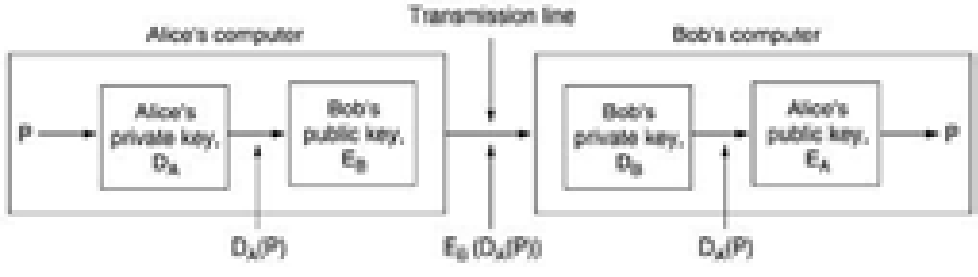


Figure 21-2. Digital signatures using public-key cryptography

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.

21.4 Problems in using public-key cryptography for digital signatures

- ✓ Bob can prove that a message was sent by Alice only as long as D_A remains secret. If Alice discloses her secret key, the argument no longer holds, because anyone could have sent the message, including Bob himself.
- ✓ The problem might arise, for example, if Bob is Alice's stockbroker. Alice tells Bob to buy a certain stock or bond. Immediately thereafter, the price drops sharply. To repudiate her message to Bob, Alice runs to the police claiming that her home was burglarized and the PC holding her key was stolen. Depending on the laws in her state or country, she may or may not be legally liable, especially if she claims not to have discovered the break-in until getting home from work, several hours later.
- ✓ Another problem with the signature scheme is what happens if Alice decides to change her key. Doing so is clearly legal, and it is probably a good idea to do so periodically. If a court case later arises, as described above, the judge will apply the current E_A to $D_A(P)$ and discover that it does not produce P. Bob will look pretty stupid at this point.

In principle, any public-key algorithm can be used for digital signatures. The usual 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 uproar. DSS was criticized for being

1. Too secret (NSA designed the protocol for using El Gamal).
2. Too slow (10 to 40 times slower than RSA for checking signatures).
3. Too new (El Gamal had not yet been thoroughly analyzed).
4. 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.

21.5 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:

1. Given P, it is easy to compute MD(P).
2. Given MD(P), it is effectively impossible to find P.
3. Given P no one can find P' such that MD (P') = MD(P).
4. 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. To see how this works, consider the signature protocol of Fig. 21-1 again. 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).

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, as shown in Fig. 21-3. 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.

Figure 21-3. Digital signatures using message digests.

21.5.1 MD5

A most widely used message digest is 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 precomputation 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 vulnerability has 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.

21.5.2 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. 21-4. 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.

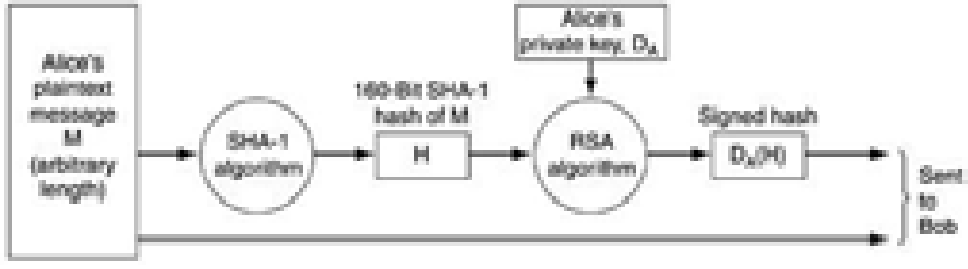


Figure 21-4. Use of SHA-1 and RSA for signing nonsecret messages.

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, the scheme of Fig. 21-4 is widely used. 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.
- ✓ In Fig. 21-5, 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.

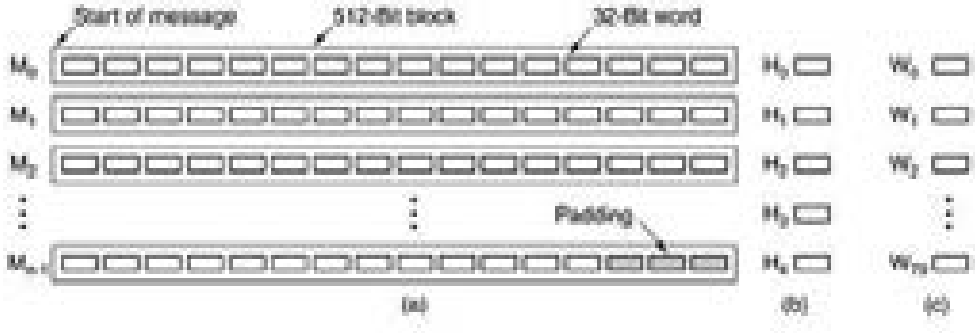


Figure 21-5. (a) A message padded out to a multiple of 512 bits. (b) The output variables. (c) The word array.

- ✓ During the computation, SHA-1 maintains five 32-bit variables, H_0 through H_4, where the hash accumulates. These are shown in Fig. 21-5(b). 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, as shown in Fig. 21-5(c).
- ✓ Then the other 64 words in W are filled in using the formula

$$W_i = S^1(W_{i-3}\ XOR\ W_{i-8}\ XOR\ W_{i-14}\ XOR\ W_{i-16}) \quad (16 \le i \le 79)$$

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 = S5(A) + fi (B, C, D) + E + Wi +Ki;
    E=D; D=C; C=S30(B); B = A; A = temp;
}
```

where the K_i constants are defined in the standard.

- ✓ The mixing functions f_i are defined as

$f_i(B,C,D) = (B \text{ AND } C) \text{ OR } (\text{NOT } B \text{ AND } D)$ $(0 \le i \le 19)$
$f_i(B,C,D) = B \text{ XOR } C \text{ XOR } D$ $(20 \le i \le 39)$
$f_i(B,C,D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D)$ $(40 \le i \le 59)$
$f_i(B,C,D) = B \text{ XOR } C \text{ XOR } D$ $(60 \le i \le 79)$

- ✓ When all 80 iterations of the loop are completed, A through E are 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. New versions of SHA-1 are under development for hashes of 256, 384, and 512 bits, respectively.

21.6 The Birthday Attack

In the world of crypto, nothing is ever what it seems to be. One might think that it would take on the order of 2^m operations to subvert an m-bit message digest. In fact, $2^{m/2}$ operations will often do using the birthday attack, an approach published by Yuval (1979) in his now-classic paper "How to Swindle Rabin."

The idea for this attack comes from a technique that math professors often use in their probability courses. The question is: How many students do you need in a class before the probability of having two people with the same birthday exceeds 1/2? Most students expect the answer to be way over 100. In fact, probability theory says it is just 23. Without giving a rigorous analysis, intuitively, with 23 people, we can form (23 x 22)/2 = 253 different pairs, each of which has a probability of 1/365 of being a hit. In this light, it is not really so surprising any more.

More generally, if there is some mapping between inputs and outputs with n inputs (people, messages, etc.) and k possible outputs (birthdays, message digests, etc.), there are n(n - 1)/2 input pairs. If n(n - 1)/2 >k, the chance of having at least one match is pretty good. Thus, approximately, a match is likely for $n>\sqrt{k}$. This result means that a 64-bit message digest can probably be broken by generating about 2^{32} messages and looking for two with the same message digest.

Let us look at a practical example. The Department of Computer Science at State University has one position for a tenured faculty member and two candidates, Tom and Dick. Tom was hired two years before Dick, so he goes up for review first. If he gets it, Dick is out of luck. Tom knows that the department chairperson, Marilyn, thinks highly of his work, so he asks her to write him a letter of recommendation to the Dean, who will decide on Tom's case. Once sent, all letters become confidential.

Marilyn tells her secretary, Ellen, to write the Dean a letter, outlining what she wants in it. When it is ready, Marilyn will review it, compute and sign

the 64-bit digest, and send it to the Dean. Ellen can send the letter later by e-mail.

Unfortunately for Tom, Ellen is romantically involved with Dick and would like to do Tom in, so she writes the letter below with the 32 bracketed options.

> Dear Dean Smith,
>
> This [letter | message] is to give my [honest | frank] opinion of Prof. Tom Wilson, who is [a candidate | up] for tenure [now | this year]. I have [known | worked with] Prof. Wilson for [about | almost] six years. He is an [outstanding | excellent] researcher of great [talent | ability] known [worldwide | internationally] for his [brilliant | creative] insights into [many | a wide variety of] [difficult | challenging] problems.
>
> He is also a [highly | greatly] [respected | admired] [teacher | educator]. His students give his [classes | courses] [rave | spectacular] reviews. He is [our | the Department's] [most popular | best-loved] [teacher | instructor].
>
> [In addition | Additionally] Prof. Wilson is a [gifted | effective] fund raiser. His [grants | contracts] have brought a [large | substantial] amount of money into [the | our] Department. [This money has | These funds have] [enabled | permitted] us to [pursue | carry out] many [special | important] programs, [such as | for example] your State 2000 program. Without these funds we would [be unable | not be able] to continue this program, which is so [important | essential] to both of us. I strongly urge you to grant him tenure.

Unfortunately for Tom, as soon as Ellen finishes composing and typing in this letter, she also writes a second one:

> Dear Dean Smith,
>
> This [letter | message] is to give my [honest | frank] opinion of Prof. Tom Wilson, who is [a candidate | up] for tenure [now | this year]. I have [known | worked with] Tom for [about | almost] six years. He is a [poor | weak] researcher not well known in his [field | area]. His research [hardly ever | rarely] shows [insight in | understanding of] the [key | major] problems of [the | our] day.
>
> Furthermore, he is not a [respected | admired] [teacher | educator]. His students give his [classes | courses] [poor | bad] reviews. He is [our | the Department's] least popular [teacher | instructor], known [mostly | primarily] within [the | our] Department for his [tendency | propensity] to [ridicule | embarrass] students [foolish | imprudent] enough to ask questions in his classes.
>
> [In addition | Additionally] Tom is a [poor | marginal] fund raiser. His [grants | contracts] have brought only a [meager | insignificant] amount of money into [the | our] Department.

> Unless new [money is | funds are] quickly located, we may have to cancel some essential programs, such as your State 2000 program. Unfortunately, under these [conditions | circumstances] I cannot in good [conscience | faith] recommend him to you for [tenure | a permanent position].

Now Ellen programs her computer to compute the 2^{32} message digests of each letter overnight. Chances are, one digest of the first letter will match one digest of the second letter. If not, she can add a few more options and try again during the weekend. Suppose that she finds a match. Call the "good" letter A and the "bad" one B.

Ellen now e-mails letter A to Marilyn for her approval. Letter B she keeps completely secret, showing it to no one. Marilyn, of course, approves, computes her 64-bit message digest, signs the digest, and e-mails the signed digest off to Dean Smith. Independently, Ellen e-mails letter B to the Dean (not letter A, as she is supposed to).

After getting the letter and signed message digest, the Dean runs the message digest algorithm on letter B, sees that it agrees with what Marilyn sent him, and fires Tom. The Dean does not realize that Ellen managed to generate two letters with the same message digest and sent her a different one than Marilyn saw and approved. (Optional ending: Ellen tells Dick what she did. Dick is appalled and breaks off with her. Ellen is furious and confesses to Marilyn. Marilyn calls the Dean. Tom gets tenure after all.) With MD5 the birthday attack is difficult because even at 1 billion digests per second, it would take over 500 years to compute all 2^{64} digests of two letters with 64 variants each, and even then a match is not guaranteed. Of course, with 5000 computers working in parallel, 500 years becomes 5 weeks. SHA-1 is better (because it is longer).

21.7 Summary

Legal, commercial, and other documents need to be signed. Accordingly, various schemes have been devised for digital signatures, using both symmetric-key and public-key algorithms. Commonly, messages to be signed are hashed using algorithms such as MD5 or SHA-1, and then the hashes are signed rather than the original messages.

Public-key management can be done using certificates, which are documents that bind a principal to a public key. Certificates are signed by a trusted authority or by someone (recursively) approved by a trusted authority. The root of the chain has to be obtained in advance, but browsers generally have many root certificates built into them.