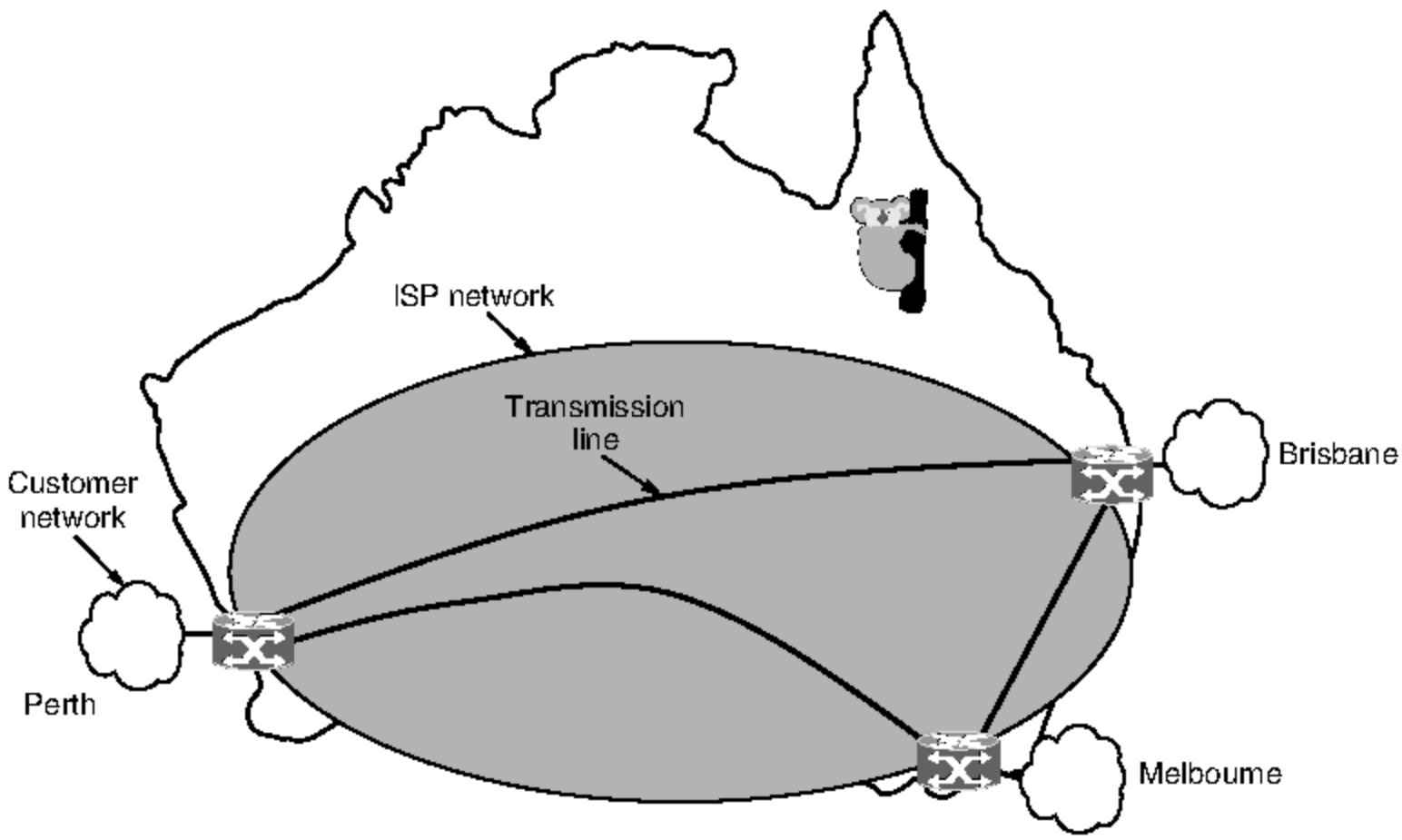

Figure 1-12. WAN using an ISP network.

may be many paths in the network that connect these two routers. How the network makes the decision as to which path to use is called the **routing algorithm**. Many such algorithms exist. How each router makes the decision as to where to send a packet next is called the **forwarding algorithm**. Many of them exist too. We will study some of both types in detail in Chap. 5.

Other kinds of WANs make heavy use of wireless technologies. In satellite systems, each computer on the ground has an antenna through which it can send data to and receive data from to a satellite in orbit. All computers can hear the output *from* the satellite, and in some cases they can also hear the upward transmissions of their fellow computers *to* the satellite as well. Satellite networks are inherently broadcast and are most useful when the broadcast property is important.

The cellular telephone network is another example of a WAN that uses wireless technology. This system has already gone through three generations and a fourth one is on the horizon. 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. Each cellular base station covers a distance much larger than a wireless LAN, with a range measured in kilometers rather than tens of meters. The base stations are connected to each other by a backbone network that is usually wired. The data rates of cellular networks are often on the order of 1 Mbps, much smaller than a wireless LAN that can range up to on the order of 100 Mbps. We will have a lot to say about these networks in Chap. 2.

1.2.5 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. 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. The Internet uses ISP networks to connect enterprise networks, home networks, and many other networks. We will look at the Internet in great detail later in this book.

Subnets, networks, and internetworks are often confused. The term "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.

A network is formed by the combination of a subnet and its hosts. However, the word "network" is often used in a loose sense as well. A subnet might be described as a network, as in the case of the "ISP network" of Fig. 1-12. An internetwork might also be described as a network, as in the case of the WAN in Fig. 1-10. We will follow similar practice, and if we are distinguishing a network from other arrangements, we will stick with our original definition of a collection of computers interconnected by a single technology.

Let us say more about what constitutes an internetwork. We know that an internet is formed when distinct networks are interconnected. In our view, connecting a LAN and a WAN or connecting two LANs is the usual way to form an internetwork, but there is little agreement in the industry over terminology in this area. There are two rules of thumb that are useful. First, if different organizations have paid to construct different parts of the network and each maintains its part, we have an internetwork rather than a single network. Second, if the underlying technology is different in different parts (e.g., broadcast versus point-to-point and wired versus wireless), we probably have an internetwork.

To go deeper, we need to talk about how two different networks can be connected. The general name for a machine that makes a connection between two or more networks and provides the necessary translation, both in terms of hardware and software, is a **gateway**. Gateways are distinguished by the layer at which they operate in the protocol hierarchy. We will have much more to say about layers and protocol hierarchies starting in the next section, but for now imagine that higher layers are more tied to applications, such as the Web, and lower layers are more tied to transmission links, such as Ethernet.

Since the benefit of forming an internet is to connect computers across networks, we do not want to use too low-level a gateway or we will be unable to make connections between different kinds of networks. We do not want to use too high-level a gateway either, or the connection will only work for particular applications. The level in the middle that is "just right" is often called the network layer, and a router is a gateway that switches packets at the network layer. We can now spot an internet by finding a network that has routers.

1.3 NETWORK SOFTWARE

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

1.3.1 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 while 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 is 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.

When 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. As an analogy, when a woman is introduced to a man, she may choose to stick out her hand. He, in turn, may decide to either shake it or kiss it, depending, for example, on whether she is an American lawyer at a business meeting or a European princess at a formal ball. Violating the protocol will make communication more difficult, if not completely impossible.

A five-layer network is illustrated in Fig. 1-13. The entities comprising the corresponding layers on different machines are called **peers**. The peers may be

software processes, hardware devices, or even human beings. In other words, it is the peers that communicate by using the protocol to talk to each other.

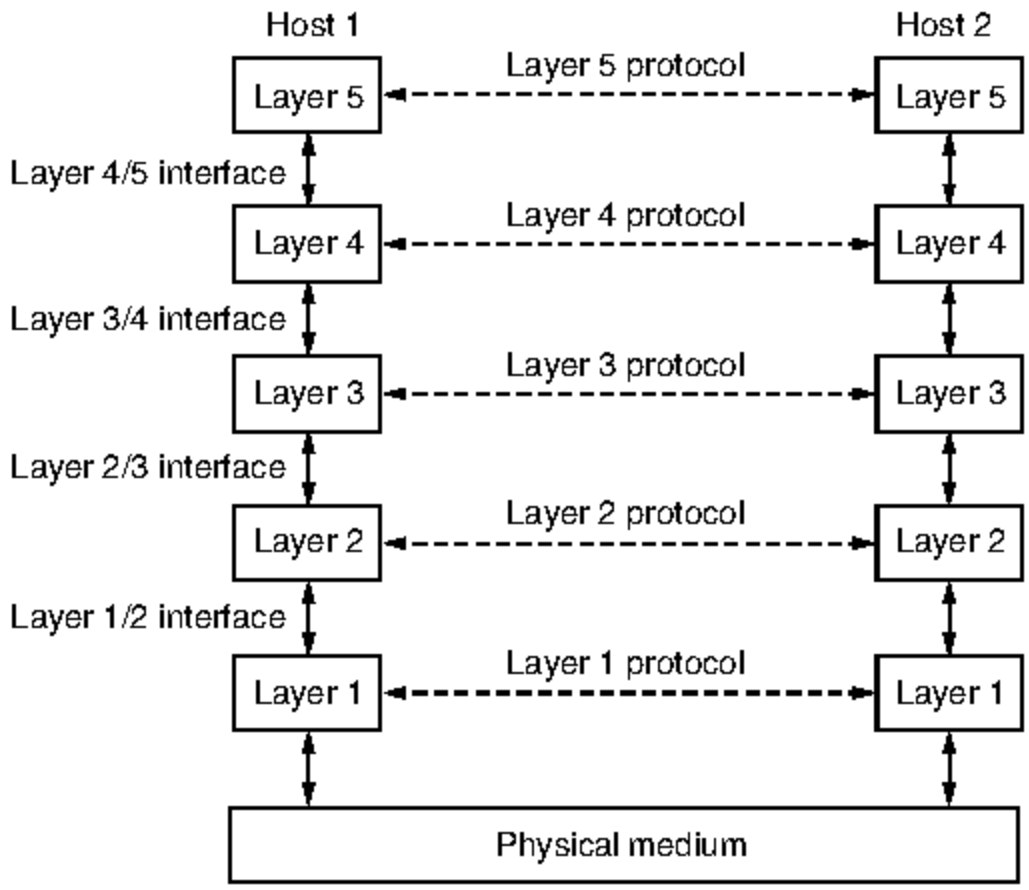

Figure 1-13. Layers, protocols, and interfaces.

In reality, no data are directly transferred from layer n on one machine to layer n on another machine. Instead, each layer passes data and control information to the layer immediately below it, until the lowest layer is reached. Below layer 1 is the **physical medium** through which actual communication occurs. In Fig. 1-13, virtual communication is shown by dotted lines and physical communication by solid lines.

Between each pair of adjacent layers is an **interface**. The interface defines which primitive operations and services the lower layer makes available to the upper one. When network designers decide how many layers to include in a network and what each one should do, one of the most important considerations is defining clean interfaces between the layers. Doing so, in turn, requires that each layer perform a specific collection of well-understood functions. In addition to minimizing the amount of information that must be passed between layers, clear-cut interfaces also make it simpler to replace one layer with a completely different protocol or implementation (e.g., replacing all the telephone lines by satellite channels) because all that is required of the new protocol or implementation is that it offer exactly the same set of services to its upstairs neighbor as the old one did. It is common that different hosts use different implementations of the same protocol (often written by different companies). In fact, the protocol itself can change in some layer without the layers above and below it even noticing.

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

An analogy may help explain the idea of multilayer communication. Imagine two philosophers (peer processes in layer 3), one of whom speaks Urdu and English and one of whom speaks Chinese and French. Since they have no common language, they each engage a translator (peer processes at layer 2), each of whom in turn contacts a secretary (peer processes in layer 1). Philosopher 1 wishes to convey his affection for *oryctolagus cuniculus* to his peer. To do so, he passes a message (in English) across the 2/3 interface to his translator, saying "I like rabbits," as illustrated in Fig. 1-14. The translators have agreed on a neutral language known to both of them, Dutch, so the message is converted to "Ik vind konijnen leuk." The choice of the language is the layer 2 protocol and is up to the layer 2 peer processes.

The translator then gives the message to a secretary for transmission, for example, by email (the layer 1 protocol). When the message arrives at the other secretary, it is passed to the local translator, who translates it into French and passes it across the 2/3 interface to the second philosopher. Note that each protocol is completely independent of the other ones as long as the interfaces are not changed. The translators can switch from Dutch to, say, Finnish, at will, provided that they both agree and neither changes his interface with either layer 1 or layer 3. Similarly, the secretaries can switch from email to telephone without disturbing (or even informing) the other layers. Each process may add some information intended only for its peer. This information is not passed up to the layer above.

Now consider a more technical example: how to provide communication to the top layer of the five-layer network in Fig. 1-15. A message, M, is produced by an application process running in layer 5 and given to layer 4 for transmission. Layer 4 puts a **header** in front of the message to identify the message and passes the result to layer 3. The header includes control information, such as addresses, to allow layer 4 on the destination machine to deliver the message. Other examples of control information used in some layers are sequence numbers (in case the lower layer does not preserve message order), sizes, and times.

In many networks, no limit is placed on the size of messages transmitted in the layer 4 protocol but there is nearly always a limit imposed by the layer 3 protocol. Consequently, layer 3 must break up the incoming messages into smaller

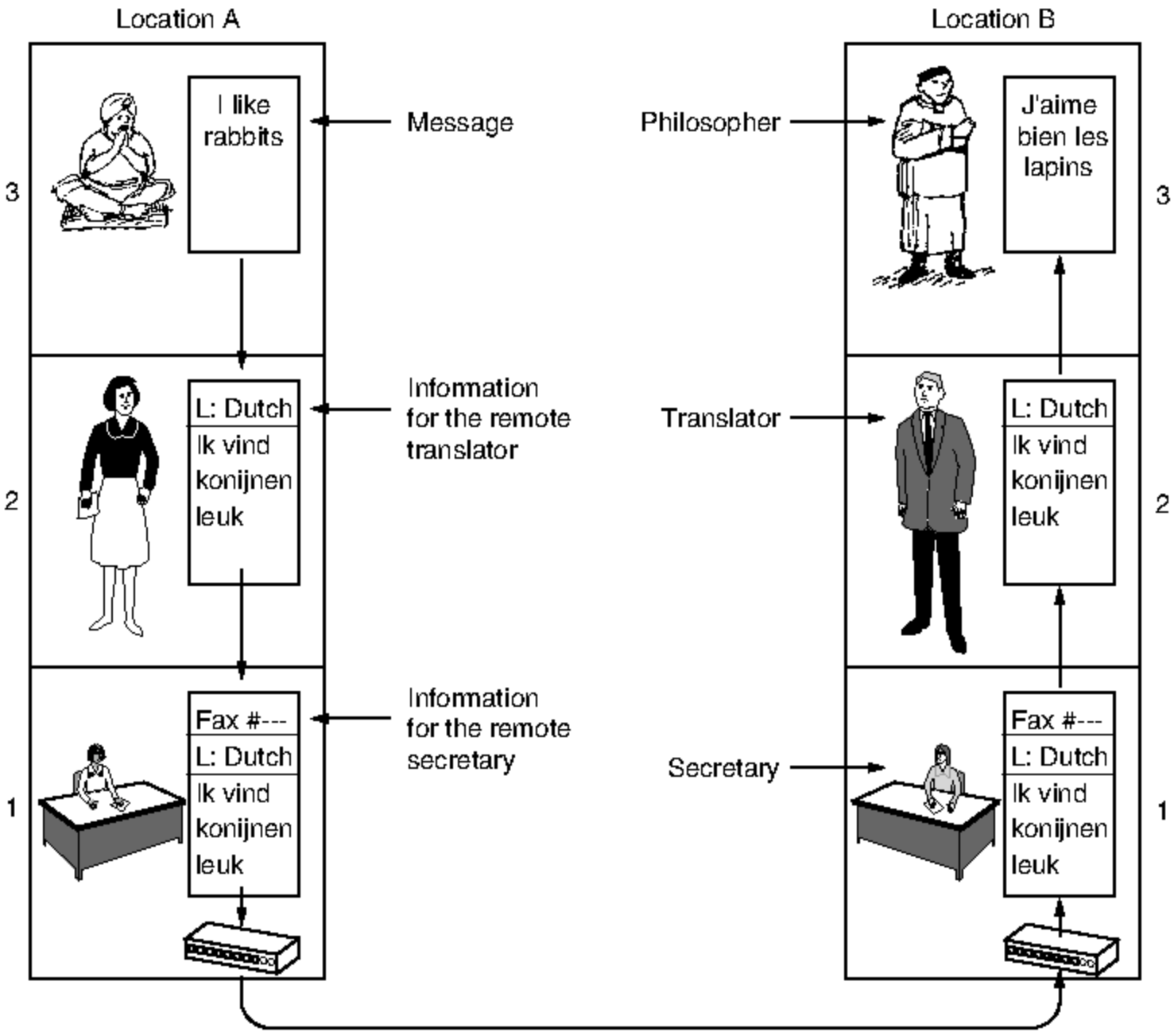

Figure 1-14. The philosopher-translator-secretary architecture.

units, packets, prepending a layer 3 header to each packet. In this example, M is split into two parts, M_1 and M_2, that will be transmitted separately.

Layer 3 decides which of the outgoing lines to use and passes the packets to layer 2. Layer 2 adds to each piece not only a header but also a trailer, and gives the resulting unit to layer 1 for physical transmission. At the receiving machine the message moves upward, from layer to layer, with headers being stripped off as it progresses. None of the headers for layers below n are passed up to layer n.

The important thing to understand about Fig. 1-15 is the relation between the virtual and actual communication and the difference between protocols and interfaces. The peer processes in layer 4, for example, conceptually think of their communication as being "horizontal," using the layer 4 protocol. Each one is likely to have procedures called something like *SendToOtherSide* and *GetFromOtherSide*, even though these procedures actually communicate with lower layers across the 3/4 interface, and not with the other side.

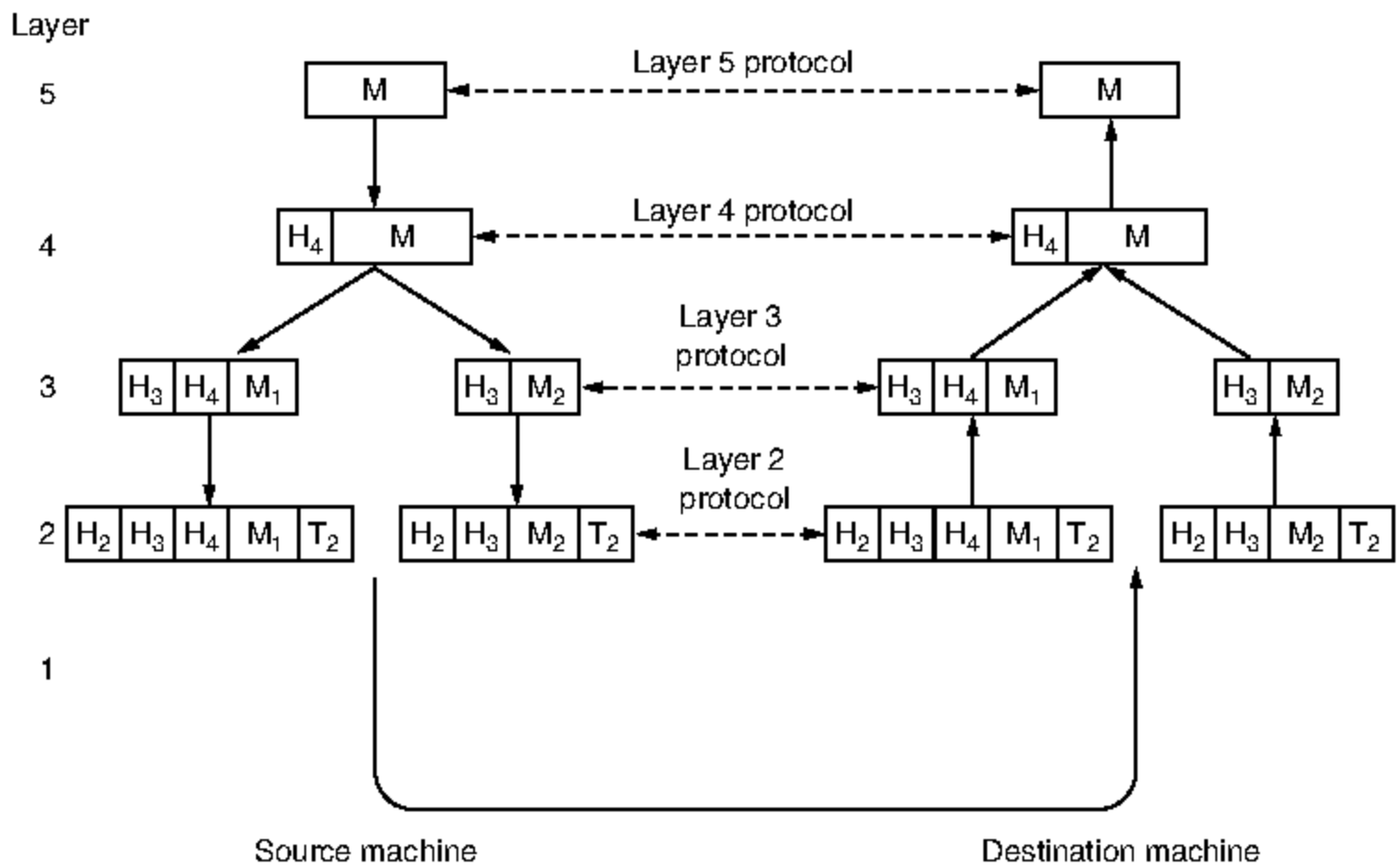

Figure 1-15. Example information flow supporting virtual communication in layer 5.

The peer process abstraction is crucial to all network design. Using it, the unmanageable task of designing the complete network can be broken into several smaller, manageable design problems, namely, the design of the individual layers.

Although Sec. 1.3 is called "Network Software," it is worth pointing out that the lower layers of a protocol hierarchy are frequently implemented in hardware or firmware. Nevertheless, complex protocol algorithms are involved, even if they are embedded (in whole or in part) in hardware.

1.3.2 Design Issues for the Layers

Some of the key design issues that occur in computer networks will come up in layer after layer. Below, we will briefly mention the more important ones.

Reliability is the design issue of making a network that operates correctly even though it is made up of a collection of components that are themselves unreliable. Think about the bits of a packet traveling through the network. There is a chance that some of these bits will be received damaged (inverted) due to fluke electrical noise, random wireless signals, hardware flaws, software bugs and so on. How is it possible that we find and fix these errors?

One mechanism for finding errors in received information uses codes for **error detection**. Information that is incorrectly received can then be retransmitted

until it is received correctly. More powerful codes allow for **error correction**, where the correct message is recovered from the possibly incorrect bits that were originally received. Both of these mechanisms work by adding redundant information. They are used at low layers, to protect packets sent over individual links, and high layers, to check that the right contents were received.

Another reliability issue is finding a working path through a network. Often there are multiple paths between a source and destination, and in a large network, there may be some links or routers that are broken. Suppose that the network is down in Germany. Packets sent from London to Rome via Germany will not get through, but we could instead send packets from London to Rome via Paris. The network should automatically make this decision. This topic is called **routing**.

A second design issue concerns the evolution of the network. Over time, networks grow larger and new designs emerge that need to be connected to the existing network. We have recently seen the key structuring mechanism used to support change by dividing the overall problem and hiding implementation details: **protocol layering**. There are many other strategies as well.

Since there are many computers on the network, every layer needs a mechanism for identifying the senders and receivers that are involved in a particular message. This mechanism is called **addressing** or **naming**, in the low and high layers, respectively.

An aspect of growth is that different network technologies often have different limitations. For example, not all communication channels preserve the order of messages sent on them, leading to solutions that number messages. Another example is differences in the maximum size of a message that the networks can transmit. This leads to mechanisms for disassembling, transmitting, and then reassembling messages. This overall topic is called **internetworking**.

When networks get large, new problems arise. Cities can have traffic jams, a shortage of telephone numbers, and it is easy to get lost. Not many people have these problems in their own neighborhood, but citywide they may be a big issue. Designs that continue to work well when the network gets large are said to be **scalable**.

A third design issue is resource allocation. Networks provide a service to hosts from their underlying resources, such as the capacity of transmission lines. To do this well, they need mechanisms that divide their resources so that one host does not interfere with another too much.

Many designs share network bandwidth dynamically, according to the short-term needs of hosts, rather than by giving each host a fixed fraction of the bandwidth that it may or may not use. This design is called **statistical multiplexing**, meaning sharing based on the statistics of demand. It can be applied at low layers for a single link, or at high layers for a network or even applications that use the network.

An allocation problem that occurs at every level is how to keep a fast sender from swamping a slow receiver with data. Feedback from the receiver to the

sender is often used. This subject is called **flow control**. Sometimes the problem is that the network is oversubscribed because too many computers want to send too much traffic, and the network cannot deliver it all. This overloading of the network is called **congestion**. One strategy is for each computer to reduce its demand when it experiences congestion. It, too, can be used in all layers.

It is interesting to observe that the network has more resources to offer than simply bandwidth. For uses such as carrying live video, the timeliness of delivery matters a great deal. Most networks must provide service to applications that want this **real-time** delivery at the same time that they provide service to applications that want high throughput. **Quality of service** is the name given to mechanisms that reconcile these competing demands.

The last major design issue is to secure the network by defending it against different kinds of threats. One of the threats we have mentioned previously is that of eavesdropping on communications. Mechanisms that provide **confidentiality** defend against this threat, and they are used in multiple layers. Mechanisms for **authentication** prevent someone from impersonating someone else. They might be used to tell fake banking Web sites from the real one, or to let the cellular network check that a call is really coming from your phone so that you will pay the bill. Other mechanisms for **integrity** prevent surreptitious changes to messages, such as altering "debit my account $10" to "debit my account $1000." All of these designs are based on cryptography, which we shall study in Chap. 8.

1.3.3 Connection-Oriented Versus Connectionless Service

Layers can offer two different types of service to the layers above them: connection-oriented and connectionless. In this section we will look at these two types and examine the differences between them.

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

In some cases when a connection is established, the sender, receiver, and subnet conduct a **negotiation** about the 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. A **circuit** is another name for a connection with associated resources, such as a fixed bandwidth. This dates from the telephone network in which a circuit was a path over copper wire that carried a phone conversation.

In contrast to connection-oriented service, **connectionless** service is modeled after the postal system. Each message (letter) carries the full destination address,

and each one is routed through the intermediate nodes inside the system independent of all the subsequent messages. There are different names for messages in different contexts; a **packet** is a message at the network layer. When the intermediate nodes receive a message in full before sending it on to the next node, this is called **store-and-forward switching**. The alternative, in which the onward transmission of a message at a node starts before it is completely received by the node, is called **cut-through switching**. 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 kind of service can further be characterized by its reliability. 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, to download a DVD movie, a byte stream from the server to the user's computer is all that is needed. Message boundaries within the movie are not relevant.

For some applications, the transit delays introduced by acknowledgements are unacceptable. One such application is digitized voice traffic for **voice over IP**. It is less disruptive 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 and starts to correct errors is irritating.

Not all applications require connections. For example, spammers send electronic junk-mail to many recipients. The spammer probably does not want to go to the trouble of setting up and later tearing down a connection to a recipient just to send them 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. Despite it being unreliable, it is the dominant form in most networks for reasons that will become clear later

In other situations, the convenience of not having to establish a connection to send one 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. Text messaging on mobile phones is an example.

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. Request-reply is commonly used to implement communication in the client-server model: the client issues a request and the server responds to it. For example, a mobile phone client might send a query to a map server to retrieve the map data for the current location. Figure 1-16 summarizes the types of services discussed above.

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Movie download
	Unreliable connection	Voice over IP
Connectionless	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Text messaging
	Request-reply	Database query

Figure 1-16. Six different types of service.

The concept of using unreliable communication may be confusing at first. After all, why would anyone actually prefer unreliable communication to reliable communication? First of all, reliable communication (in our sense, that is, acknowledged) may not be available in a given layer. For example, Ethernet does not provide reliable communication. Packets can occasionally be damaged in transit. It is up to higher protocol levels to recover from this problem. In particular, many reliable services are built on top of an unreliable datagram service. Second, the delays inherent in providing a reliable service may be unacceptable, especially in real-time applications such as multimedia. For these reasons, both reliable and unreliable communication coexist.

1.3.4 Service Primitives

A service is formally specified by a set of **primitives** (operations) available to user processes 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 provide a reliable byte stream, consider the primitives listed in Fig. 1-17. They will be familiar to fans of the Berkeley socket interface, as the primitives are a simplified version of that interface.

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

Figure 1-17. Six service primitives that provide a simple connection-oriented service.

These primitives might be used for a request-reply interaction in a client-server environment. To illustrate how, We sketch a simple protocol that implements the service using acknowledged datagrams.

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. 1-18. The client process is suspended until there is a response.

When the packet arrives at the server, the operating system sees that the packet is requesting a connection. It checks to see if there is a listener, and if so it unblocks the listener. The server process can then establish the connection with the ACCEPT call. This sends a response (2) back to the client process to accept the

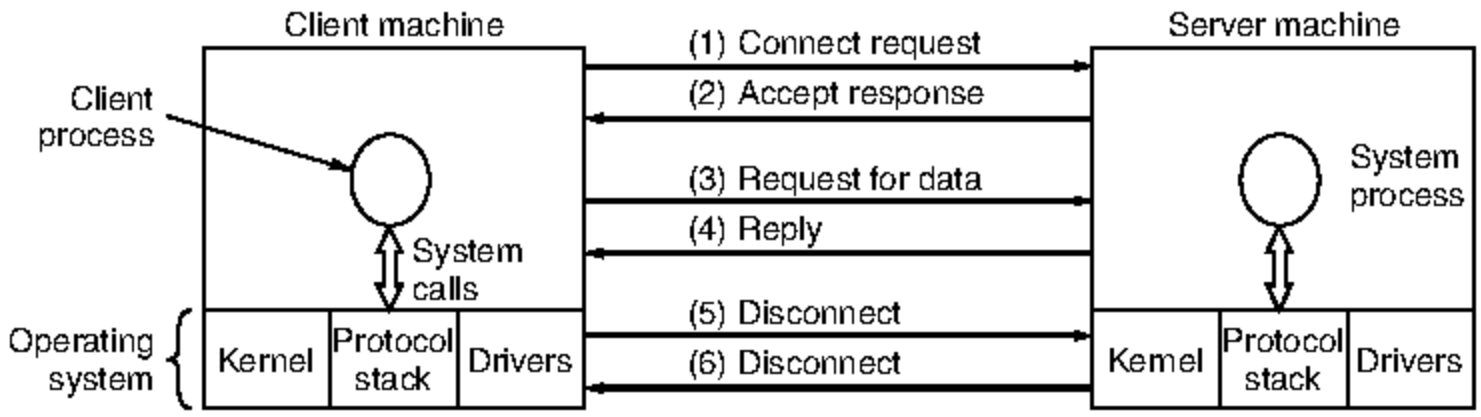

Figure 1-18. A simple client-server interaction using acknowledged datagrams.

connection. The arrival of this response then releases the client. At this point the client and server are both running and they have a connection established.

The obvious analogy between this protocol and real life is a customer (client) calling a company's customer service manager. At the start of the day, the service manager sits next to his telephone in case it rings. Later, a client places a 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 so it can handle the request. After it has done the work, the server 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.

When the client is done, it executes DISCONNECT to terminate the connection (5). 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. When the server gets the packet, it also issues a DISCONNECT of its own, acknowledging the client and releasing the connection (6). When the server's packet 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.

Of course, life is not so simple. Many things can go wrong here. 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. 1-18 briefly summarizes how client-server communication might work with acknowledged datagrams so that we can ignore lost packets.

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 either 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 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 Chap. 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.

1.3.5 The Relationship of Services to Protocols

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

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. This is a key concept that any network designer should understand well.

To repeat this crucial point, services relate to the interfaces between layers, as illustrated in Fig. 1-19. In contrast, protocols relate to the packets sent between peer entities on different machines. It is very important not to confuse the two concepts.

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

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

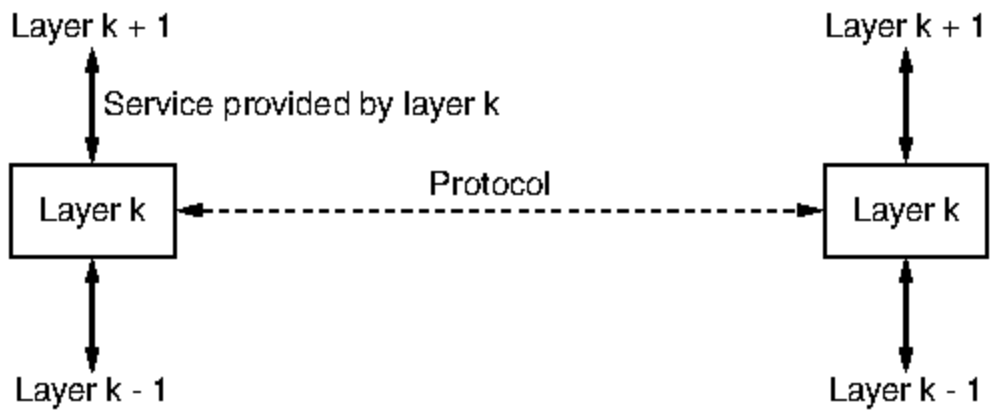

Figure 1-19. The relationship between a service and a protocol.

1.4 REFERENCE MODELS

Now that we have discussed layered networks in the abstract, it is time to look at some examples. 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 not 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.

1.4.1 The OSI Reference Model

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

The OSI 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.

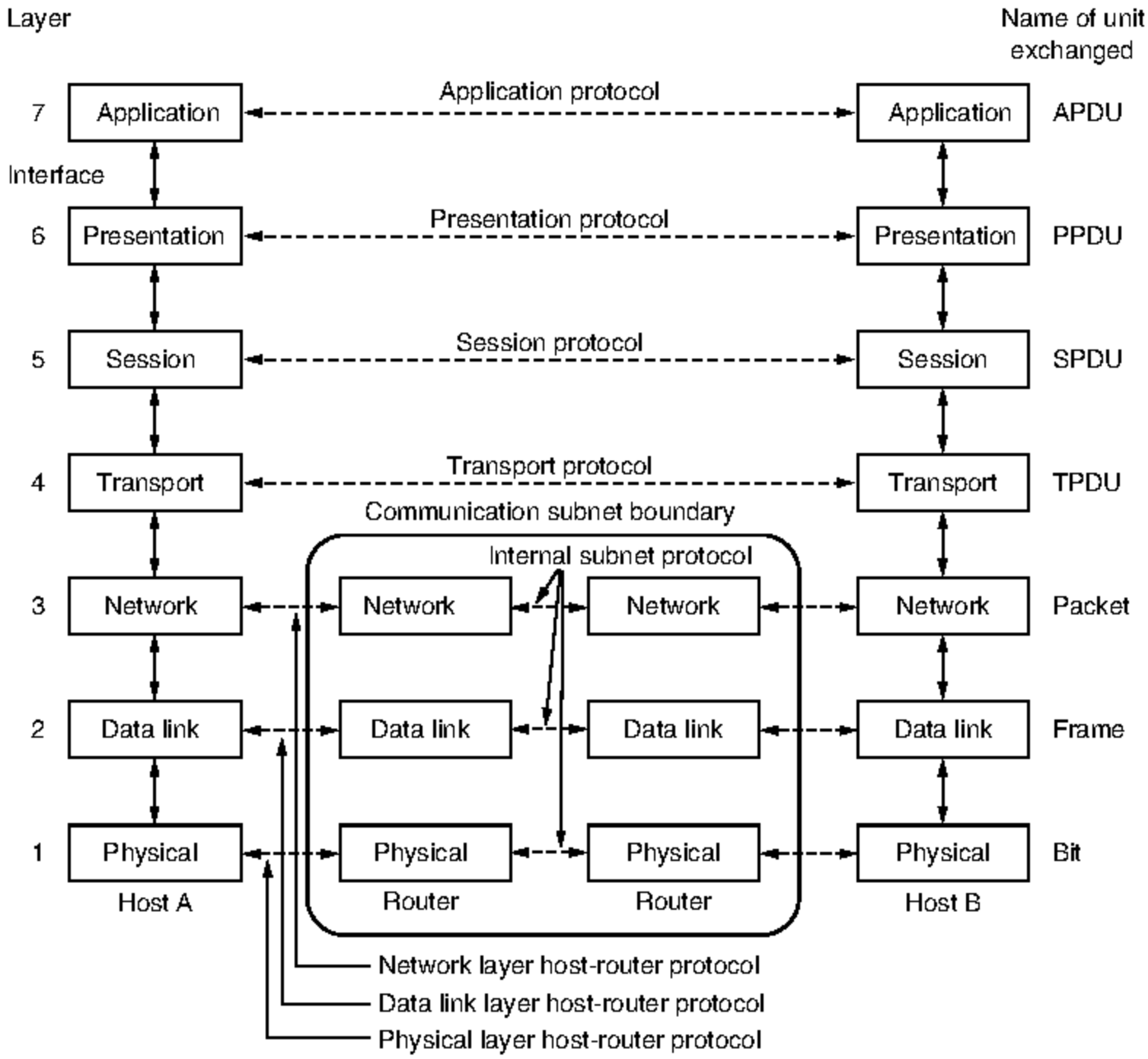

Figure 1-20. The OSI reference model.

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 a network architecture because it does not specify the exact services and protocols to be used in each layer. It just tells what each layer should do. However, ISO has also produced standards for all the layers, although these are not part of the reference model itself. Each one has been published as a separate international standard. The *model* (in part) is widely used although the associated protocols have been long forgotten.

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 what electrical signals should be used to represent a 1 and a 0, how many nanoseconds a bit lasts, whether transmission may proceed simultaneously in both directions, how the initial connection is established, how it is torn down when both sides are finished, how many pins the network connector has, and what each pin is used for. These design issues largely deal with mechanical, electrical, and timing interfaces, as well as the physical transmission medium, which lies below the physical layer.

The Data Link Layer

The main task of the **data link layer** is to transform a raw transmission facility into a line that appears free of undetected transmission errors. It does so by masking the real errors so the network layer does not see them. It accomplishes this task by having the sender break up the input data into **data frames** (typically a few hundred or a few thousand bytes) and transmit the frames sequentially. If the service is reliable, the receiver confirms correct receipt of each frame by sending back an **acknowledgement frame**.

Another issue that arises in the data link layer (and most of the higher layers as well) is how to keep a fast transmitter from drowning a slow receiver in data. Some traffic regulation mechanism may be needed to let the transmitter know when the receiver can accept more data.

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

The Network Layer

The **network layer** controls the operation of the subnet. A key design issue is determining how packets are routed from source to destination. Routes can be based on static tables that are "wired into" the network and rarely changed, or more often they can be updated automatically to avoid failed components. They can also be determined at the start of each conversation, for example, a terminal session, such as 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. Handling congestion is also a responsibility of the network layer, in conjunction with higher layers that adapt the load

they place on the network. 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 that used by 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.

The Transport Layer

The basic function of the **transport layer** is to accept data from above it, 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 over the course of time.

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 exist, such as 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 completely 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; it carries data 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, each protocols is between a 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. 1-20.

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 simultaneously), and **synchronization**

(checkpointing long transmissions to allow them to pick up from where they left off in the event of a crash and subsequent recovery).

The Presentation Layer

Unlike the 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 internal data representations to communicate, the data structures to be exchanged can be defined in an abstract way, along with a standard encoding to be used "on the wire." The presentation layer manages these abstract data structures and allows higher-level data structures (e.g., banking records) to be defined and exchanged.

The Application Layer

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

1.4.2 The TCP/IP Reference Model

Let us now turn from the OSI reference model to the reference model used in the grandparent of all wide area computer networks, the ARPANET, and its successor, the worldwide Internet. Although we will give a brief history of the ARPANET later, it is useful to mention a few key aspects of it now. The ARPANET was a research network sponsored by the DoD (U.S. Department of Defense). It eventually connected hundreds of universities and government installations, using leased telephone lines. When satellite and radio networks were added later, the existing protocols had trouble interworking with them, so a new reference architecture was needed. Thus, from nearly the beginning, the ability to connect multiple networks in a seamless way was one of the major design goals. This architecture later became known as the **TCP/IP Reference Model**, after its two primary protocols. It was first described by Cerf and Kahn (1974), and later refined and defined as a standard in the Internet community (Braden, 1989). The design philosophy behind the model is discussed by 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 by an attack from the Soviet Union, another major goal was that the network be able to survive loss of subnet hardware, without existing conversations being broken off. In other

words, the 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, since applications with divergent requirements were envisioned, ranging from transferring files to real-time speech transmission, a flexible architecture was needed.

The Link Layer

All these requirements led to the choice of a packet-switching network based on a connectionless layer that runs across different networks. The lowest layer in the model, the **link layer** describes what links such as serial lines and classic Ethernet must do to meet the needs of this connectionless internet layer. It is not really a layer at all, in the normal sense of the term, but rather an interface between hosts and transmission links. Early material on the TCP/IP model has little to say about it.

The Internet Layer

The **internet layer** is the linchpin that holds the whole architecture together. It is shown in Fig. 1-21 as corresponding roughly to the OSI network layer. 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 completely 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.

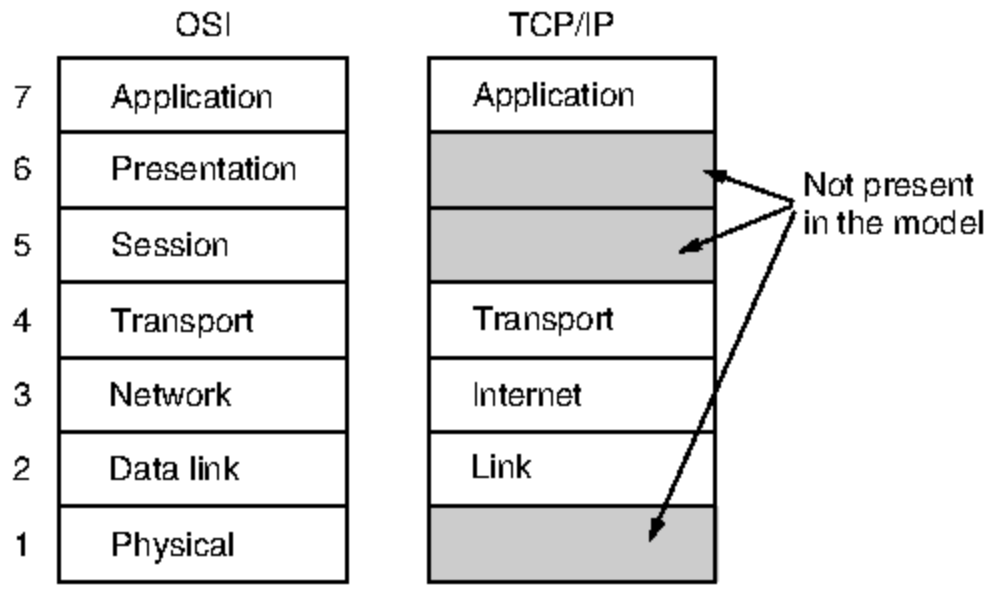

Figure 1-21. The TCP/IP reference model.

The analogy here is with the (snail) mail system. A person can drop a sequence of international letters into a mailbox in one country, and with a little luck,

most of them will be delivered to the correct address in the destination country. The letters will probably 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**), plus a companion protocol called **ICMP** (**Internet Control Message Protocol**) that helps it function. The job of the internet layer is to deliver IP packets where they are supposed to go. Packet routing is clearly a major issue here, as is congestion (though IP has not proven effective at avoiding congestion).

The Transport Layer

The layer above the internet layer in the TCP/IP model is now usually called the **transport layer**. It is designed to allow peer entities on the source and destination hosts to carry on a conversation, just as in the OSI transport layer. Two end-to-end transport protocols have been defined here. The first one, **TCP** (**Transmission Control Protocol**), is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error on any other machine in the internet. It segments 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.

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

The Application Layer

The TCP/IP model does not have session or presentation layers. No need for them was perceived. Instead, applications simply include any session and presentation functions that they require. Experience with the OSI model has proven this view correct: these layers are of little use to most applications.

On top of the transport layer is the **application layer**. It contains all the higher-level protocols. The early ones included virtual terminal (TELNET), file transfer (FTP), and electronic mail (SMTP). Many other protocols have been added to these over the years. Some important ones that we will study, shown in Fig. 1-22,

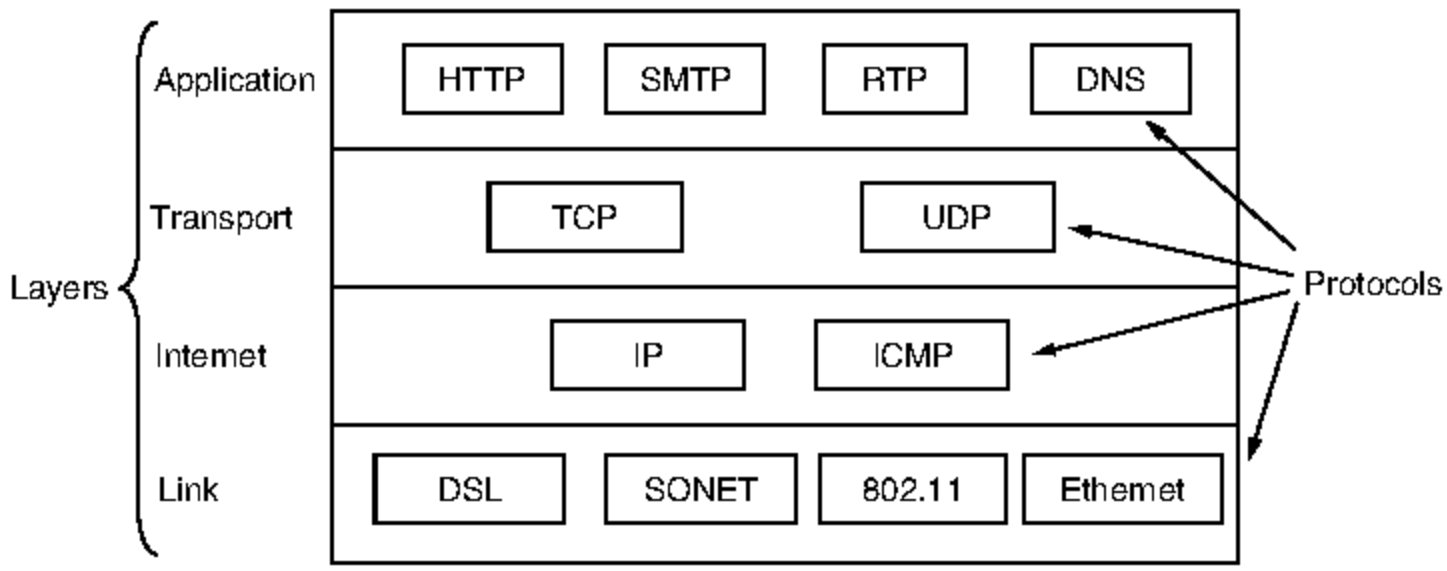

Figure 1-22. The TCP/IP model with some protocols we will study.

include the Domain Name System (DNS), for mapping host names onto their network addresses, HTTP, the protocol for fetching pages on the World Wide Web, and RTP, the protocol for delivering real-time media such as voice or movies.

1.4.3 The Model Used in This Book

As mentioned earlier, the strength of the OSI reference model is the *model* itself (minus the presentation and session layers), which has proven to be exceptionally useful for discussing computer networks. In contrast, the strength of the TCP/IP reference model is the *protocols*, which have been widely used for many years. Since computer scientists like to have their cake and eat it, too, we will use the hybrid model of Fig. 1-23 as the framework for this book.

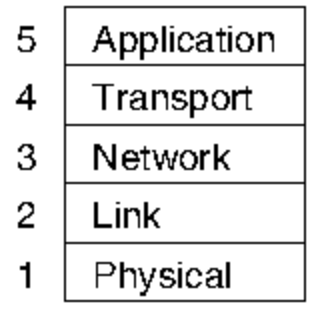

Figure 1-23. The reference model used in this book.

This model has five layers, running from the physical layer up through the link, network and transport layers to the application layer. The physical layer specifies how to transmit bits across different kinds of media as electrical (or other analog) signals. The link layer is concerned with how to send finite-length messages between directly connected computers with specified levels of reliability. Ethernet and 802.11 are examples of link layer protocols.

The network layer deals with how to combine multiple links into networks, and networks of networks, into internetworks so that we can send packets between distant computers. This includes the task of finding the path along which to send the packets. IP is the main example protocol we will study for this layer. The transport layer strengthens the delivery guarantees of the Network layer, usually with increased reliability, and provide delivery abstractions, such as a reliable byte stream, that match the needs of different applications. TCP is an important example of a transport layer protocol.

Finally, the application layer contains programs that make use of the network. Many, but not all, networked applications have user interfaces, such as a Web browser. Our concern, however, is with the portion of the program that uses the network. This is the HTTP protocol in the case of the Web browser. There are also important support programs in the application layer, such as the DNS, that are used by many applications.

Our chapter sequence is based on this model. In this way, we retain the value of the OSI model for understanding network architectures, but concentrate primarily on protocols that are important in practice, from TCP/IP and related protocols to newer ones such as 802.11, SONET, and Bluetooth.

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

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

Despite these fundamental similarities, the two models also have many differences. In this section we will focus on the key differences between the two reference models. It is important to note that we are comparing the *reference models* here, not the corresponding *protocol stacks*. The protocols themselves will be discussed later. For an entire book comparing and contrasting TCP/IP and OSI, see Piscitello and Chapin (1993).

Three concepts are central to the OSI model:

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

Probably the biggest contribution of the OSI model is that it makes the distinction between these three concepts explicit. Each layer performs some *services* for the

layer above it. The service definition tells what the layer does, not how entities above it access it or how the layer works. It defines the layer's semantics.

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

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

These ideas fit very nicely with modern ideas about object-oriented programming. An object, like a layer, has a set of methods (operations) that processes outside the object can invoke. The semantics of these methods define the set of services that the object offers. The methods' parameters and results 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.

The TCP/IP model did not originally clearly distinguish between services, interfaces, and protocols, 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 transparently is one of the main purposes of having layered protocols in the first place.

The OSI reference model was devised *before* the corresponding protocols were invented. This ordering meant that the model was not biased toward one particular set of protocols, a fact that made it quite general. The downside of this ordering was that the designers did not have much experience with the subject and did not have a good idea of which functionality to put in which layer.

For example, the data link layer originally dealt only with point-to-point networks. When broadcast networks came around, a new sublayer had to be hacked into the model. Furthermore, when people started to build real networks using the OSI model and existing protocols, it was discovered that these networks did not match the required service specifications (wonder of wonders), so convergence sublayers had to be grafted onto the model to provide a place for papering over the differences. Finally, the committee originally expected that each country would have one network, run by the government and using the OSI protocols, so no thought was given to internetworking. To make a long story short, things did not turn out that way.

With TCP/IP the reverse was true: the protocols came first, and the model was really just a description of the existing protocols. There was no problem with the protocols fitting the model. They fit perfectly. The only trouble was that the *model* did not fit any other protocol stacks. Consequently, it was not especially useful for describing other, non-TCP/IP networks.

Turning from philosophical matters to more specific ones, an obvious difference between the two models is the number of layers: the OSI model has seven layers and the TCP/IP model has four. 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 supports only one mode in the network layer (connectionless) but both in the transport layer, giving the users a choice. This choice is especially important for simple request-response protocols.

1.4.5 A Critique of the OSI Model and Protocols

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

At the time the second edition of this book was published (1989), it appeared to many experts in the field that the OSI model and its protocols were going to take over the world and push everything else out of their way. This did not happen. Why? A look back at some of the reasons may be useful. They can be summarized as:

1. Bad timing.
2. Bad technology.
3. Bad implementations.
4. Bad politics.

Bad Timing

First let us look at reason one: bad timing. The time at which a standard is established is absolutely critical to its success. David Clark of M.I.T. has a theory of standards that he calls the *apocalypse of the two elephants*, which is illustrated in Fig. 1-24.

This figure shows the amount of activity surrounding a new subject. When the subject is first discovered, there is a burst of research activity in the form of discussions, papers, and meetings. After a while this activity subsides, corporations discover the subject, and the billion-dollar wave of investment hits.

It is essential that the standards be written in the trough in between the two "elephants." If they are written too early (before the research results are well

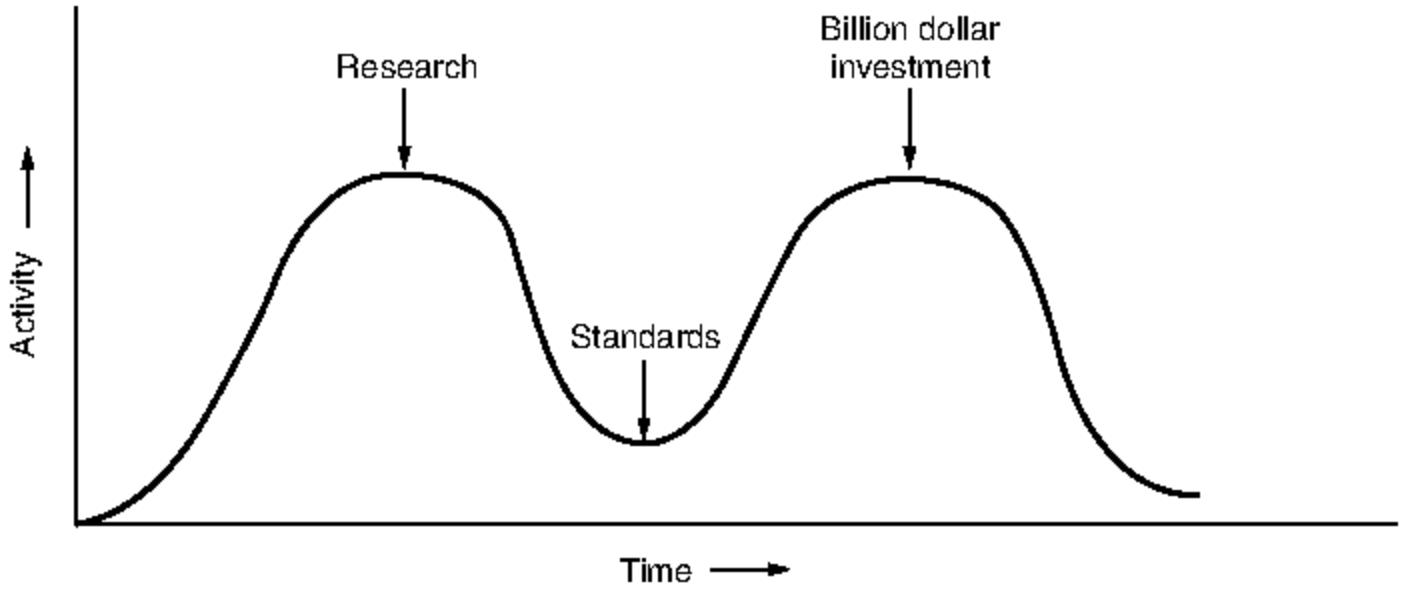

Figure 1-24. The apocalypse of the two elephants.

established), the subject may still be poorly understood; the result is a bad standard. If they are written too late, so many companies may have already made major investments in different ways of doing things that the standards are effectively ignored. If the interval between the two elephants is very short (because everyone is in a hurry to get started), the people developing the standards may get crushed.

It now appears that the standard OSI protocols got crushed. The competing TCP/IP protocols were already in widespread use by research universities by the time the OSI protocols appeared. While the billion-dollar wave of investment had not yet hit, the academic market was large enough that many vendors had begun cautiously offering TCP/IP products. When OSI came around, they did not want to support a second protocol stack until they were forced to, so there were no initial offerings. With every company waiting for every other company to go first, no company went first and OSI never happened.

Bad Technology

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

The OSI model, along with its 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 this context, a riddle posed by Paul Mockapetris and cited by Rose (1993) comes to mind:

> Q: What do you get when you cross a mobster with an international standard?
> A: Someone who makes you an offer you can't understand.

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.

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.

Bad Politics

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

OSI, on the other hand, was widely thought to be the creature of the European telecommunication ministries, the European Community, and later the U.S. Government. This belief was only partly true, but the very idea of a bunch of government bureaucrats trying to shove a technically inferior standard down the throats of the poor researchers and programmers down in the trenches actually developing computer networks did not aid OSI's cause. Some people viewed this development in the same light as IBM announcing in the 1960s that PL/I was the language of the future, or the DoD correcting this later by announcing that it was actually Ada.

1.4.6 A Critique of the TCP/IP Reference Model

The TCP/IP model and protocols have their problems too. First, the model does not clearly distinguish the concepts of services, interfaces, and protocols. Good software engineering practice requires differentiating between the specification and the implementation, something that OSI does very carefully, but 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 link 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 between 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, it is still in use some 30 years later.

1.5 EXAMPLE NETWORKS

The subject of computer networking covers many different kinds of networks, large and small, well known and less well known. They have different goals, scales, and technologies. In the following sections, we will look at some examples, to get an idea of the variety one finds in the area of computer networking.

We will start with the Internet, probably the best known network, and look at its history, evolution, and technology. Then we will consider the mobile phone network. Technically, it is quite different from the Internet, contrasting nicely with it. Next we will introduce IEEE 802.11, the dominant standard for wireless LANs. Finally, we will look at RFID and sensor networks, technologies that extend the reach of the network to include the physical world and everyday objects.

1.5.1 The Internet

The Internet is not really a network at all, but a vast collection of different networks that use certain common protocols and provide certain common services. It is an unusual system in that it was not planned by anyone and is not controlled by anyone. To better understand it, let us start from the beginning and see how it has developed and why. For a wonderful history of the Internet, John Naughton's (2000) book is highly recommended. It is one of those rare books that is not only fun to read, but also has 20 pages of *ibid.*'s and *op. cit.*'s for the serious historian. Some of the material in this section is based on this book.

Of course, countless technical books have been written about the Internet and its protocols as well. For more information, see, for example, Maufer (1999).

The ARPANET

The story begins in the late 1950s. At the height of the Cold War, the U.S. DoD wanted a command-and-control network that could survive a nuclear war. At that time, all military communications used the public telephone network, which was considered vulnerable. The reason for this belief can be gleaned from Fig. 1-25(a). Here the black dots represent telephone switching offices, each of which was connected to thousands of telephones. These switching offices were, in turn, connected to higher-level switching offices (toll offices), to form a national hierarchy with only a small amount of redundancy. The vulnerability of the system was that the destruction of a few key toll offices could fragment it into many isolated islands.

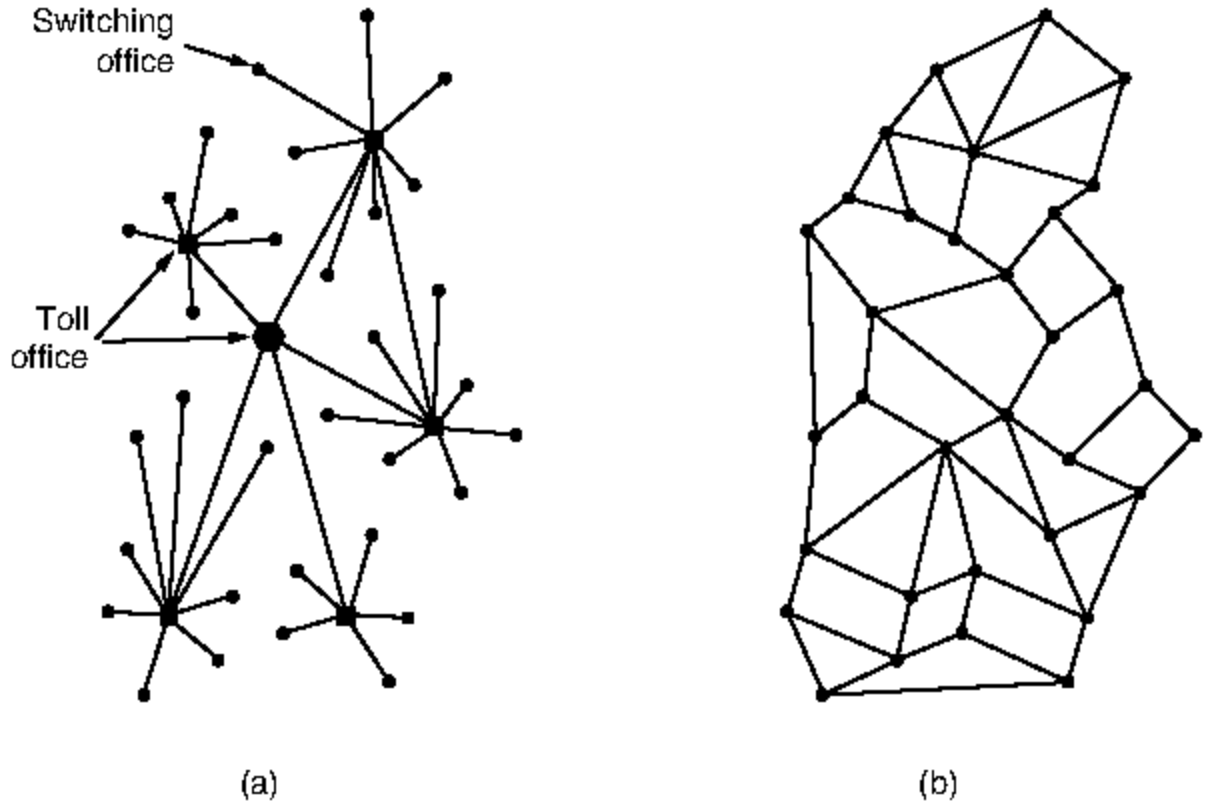

Figure 1-25. (a) Structure of the telephone system. (b) Baran's proposed distributed switching system.

Around 1960, the DoD awarded a contract to the RAND Corporation to find a solution. One of its employees, Paul Baran, came up with the highly distributed and fault-tolerant design of Fig. 1-25(b). Since the paths between any two switching offices were now much longer than analog signals could travel without distortion, Baran proposed using digital packet-switching technology. Baran wrote several reports for the DoD describing his ideas in detail (Baran, 1964). Officials at the Pentagon liked the concept and asked AT&T, then the U.S.' national telephone monopoly, to build a prototype. AT&T dismissed Baran's ideas out of hand. The biggest and richest corporation in the world was not about to allow

some young whippersnapper tell it how to build a telephone system. They said Baran's network could not be built and the idea was killed.

Several years went by and still the DoD did not have a better command-and-control system. To understand what happened next, we have to go back all the way to October 1957, when the Soviet Union beat the U.S. into space with the launch of the first artificial satellite, Sputnik. When President Eisenhower tried to find out who was asleep at the switch, he was appalled to find the Army, Navy, and Air Force squabbling over the Pentagon's research budget. His immediate response was to create a single defense research organization, **ARPA**, the **Advanced Research Projects Agency**. ARPA had no scientists or laboratories; in fact, it had nothing more than an office and a small (by Pentagon standards) budget. It did its work by issuing grants and contracts to universities and companies whose ideas looked promising to it.

For the first few years, ARPA tried to figure out what its mission should be. In 1967, the attention of Larry Roberts, a program manager at ARPA who was trying to figure out how to provide remote access to computers, turned to networking. He contacted various experts to decide what to do. One of them, Wesley Clark, suggested building a packet-switched subnet, connecting each host to its own router.

After some initial skepticism, Roberts bought the idea and presented a somewhat vague paper about it at the ACM SIGOPS Symposium on Operating System Principles held in Gatlinburg, Tennessee in late 1967 (Roberts, 1967). Much to Roberts' surprise, another paper at the conference described a similar system that had not only been designed but actually fully implemented under the direction of Donald Davies at the National Physical Laboratory in England. The NPL system was not a national system (it just connected several computers on the NPL campus), but it demonstrated that packet switching could be made to work. Furthermore, it cited Baran's now discarded earlier work. Roberts came away from Gatlinburg determined to build what later became known as the **ARPANET**.

The subnet would consist of minicomputers called **IMPs** (**Interface Message Processors**) connected by 56-kbps transmission lines. For high reliability, each IMP would be connected to at least two other IMPs. The subnet was to be a datagram subnet, so if some lines and IMPs were destroyed, messages could be automatically rerouted along alternative paths.

Each node of the network was to consist of an IMP and a host, in the same room, connected by a short wire. A host could send messages of up to 8063 bits to its IMP, which would then break these up into packets of at most 1008 bits and forward them independently toward the destination. Each packet was received in its entirety before being forwarded, so the subnet was the first electronic store-and-forward packet-switching network.

ARPA then put out a tender for building the subnet. Twelve companies bid for it. After evaluating all the proposals, ARPA selected BBN, a consulting firm based in Cambridge, Massachusetts, and in December 1968 awarded it a contract

to build the subnet and write the subnet software. BBN chose to use specially modified Honeywell DDP-316 minicomputers with 12K 16-bit words of core memory as the IMPs. The IMPs did not have disks, since moving parts were considered unreliable. The IMPs were interconnected by 56-kbps lines leased from telephone companies. Although 56 kbps is now the choice of teenagers who cannot afford DSL or cable, it was then the best money could buy.

The software was split into two parts: subnet and host. The subnet software consisted of the IMP end of the host-IMP connection, the IMP-IMP protocol, and a source IMP to destination IMP protocol designed to improve reliability. The original ARPANET design is shown in Fig. 1-26.

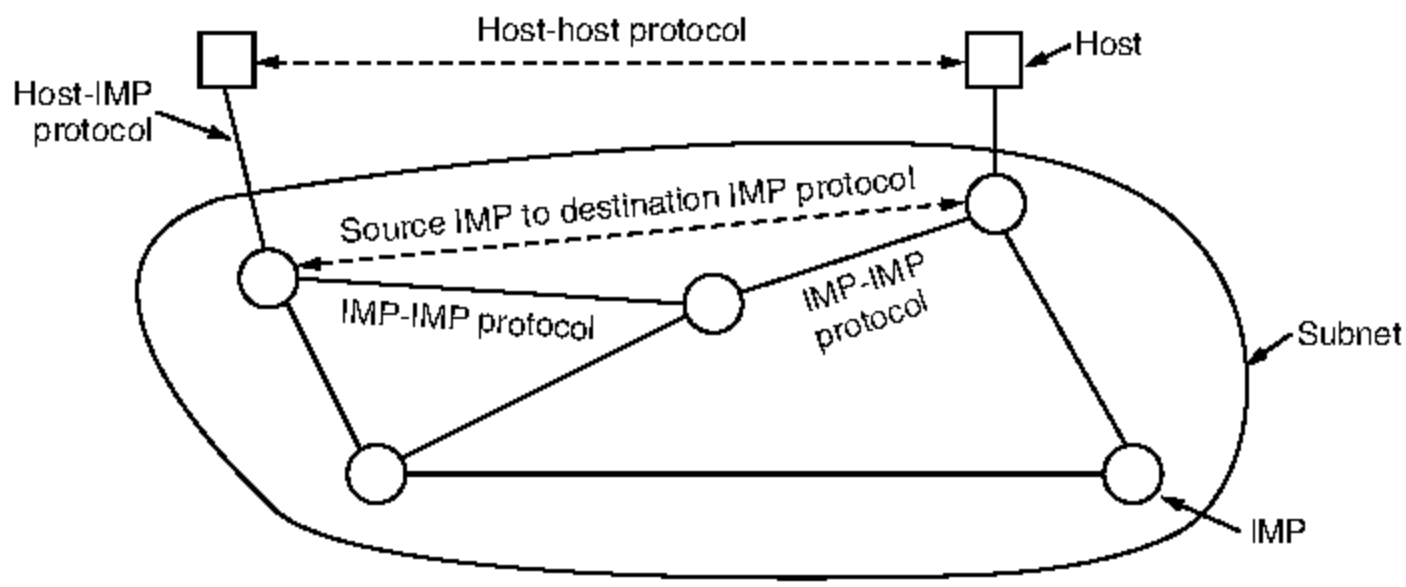

Figure 1-26. The original ARPANET design.

Outside the subnet, software was also needed, namely, the host end of the host-IMP connection, the host-host protocol, and the application software. It soon became clear that BBN was of the opinion that when it had accepted a message on a host-IMP wire and placed it on the host-IMP wire at the destination, its job was done.

Roberts had a problem, though: the hosts needed software too. To deal with it, he convened a meeting of network researchers, mostly graduate students, at Snowbird, Utah, in the summer of 1969. The graduate students expected some network expert to explain the grand design of the network and its software to them and then assign each of them the job of writing part of it. They were astounded when there was no network expert and no grand design. They had to figure out what to do on their own.

Nevertheless, somehow an experimental network went online in December 1969 with four nodes: at UCLA, UCSB, SRI, and the University of Utah. These four were chosen because all had a large number of ARPA contracts, and all had different and completely incompatible host computers (just to make it more fun). The first host-to-host message had been sent two months earlier from the UCLA

node by a team led by Len Kleinrock (a pioneer of the theory of packet switching) to the SRI node. The network grew quickly as more IMPs were delivered and installed; it soon spanned the United States. Figure 1-27 shows how rapidly the ARPANET grew in the first 3 years.

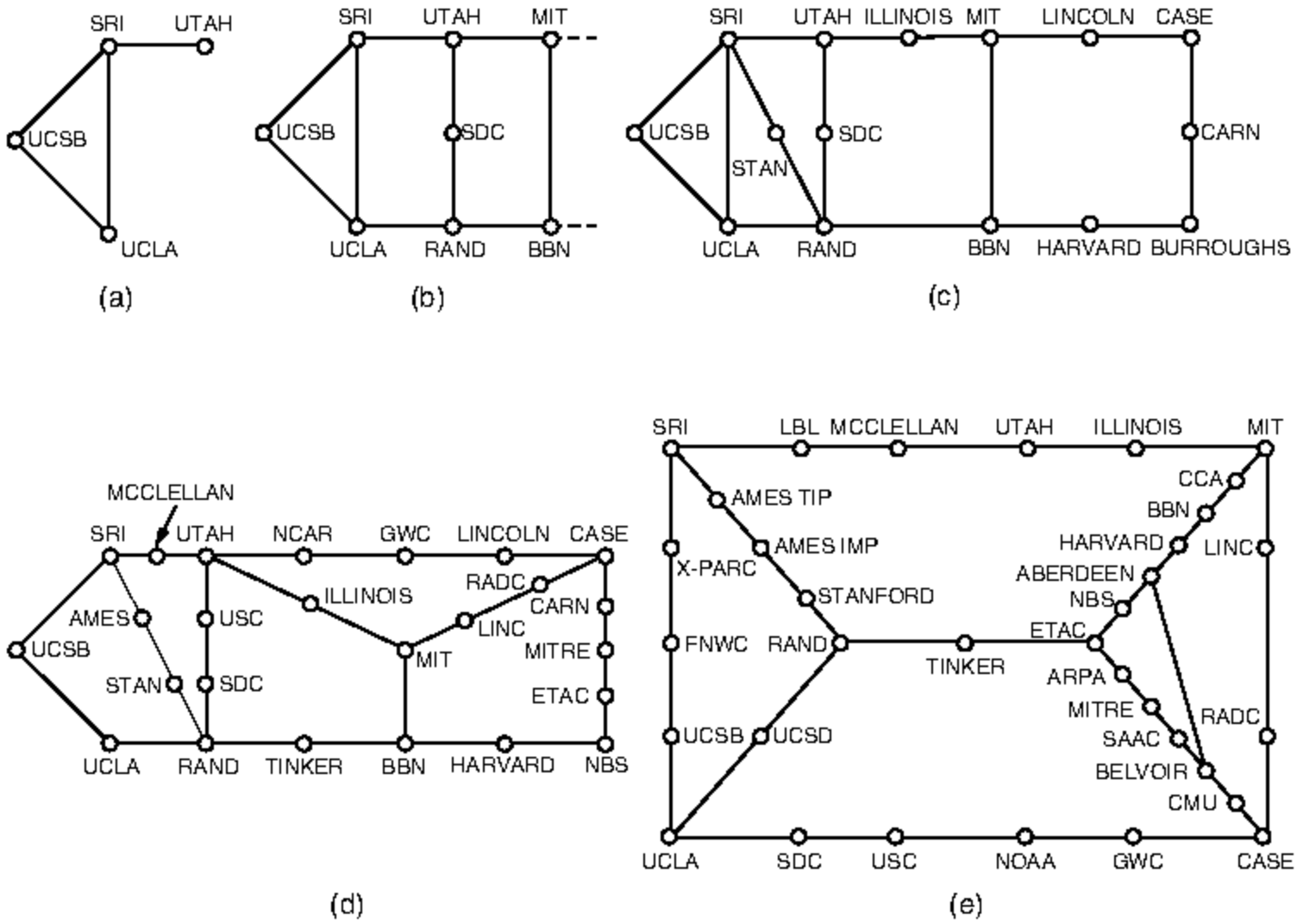

Figure 1-27. Growth of the ARPANET. (a) December 1969. (b) July 1970. (c) March 1971. (d) April 1972. (e) September 1972.

In addition to helping the fledgling ARPANET grow, ARPA also funded research on the use of satellite networks and mobile packet radio networks. In one now famous demonstration, a truck driving around in California used the packet radio network to send messages to SRI, which were then forwarded over the ARPANET to the East Coast, where they were shipped to University College in London over the satellite network. This allowed a researcher in the truck to use a computer in London while driving around in California.

This experiment also demonstrated that the existing ARPANET protocols were not suitable for running over different networks. This observation led to more research on protocols, culminating with the invention of the TCP/IP model and protocols (Cerf and Kahn, 1974). TCP/IP was specifically designed to handle communication over internetworks, something becoming increasingly important as more and more networks were hooked up to the ARPANET.

To encourage adoption of these new protocols, ARPA awarded several contracts to implement TCP/IP on different computer platforms, including IBM, DEC, and HP systems, as well as for Berkeley UNIX. Researchers at the University of California at Berkeley rewrote TCP/IP with a new programming interface called **sockets** for the upcoming 4.2BSD release of Berkeley UNIX. They also wrote many application, utility, and management programs to show how convenient it was to use the network with sockets.

The timing was perfect. Many universities had just acquired a second or third VAX computer and a LAN to connect them, but they had no networking software. When 4.2BSD came along, with TCP/IP, sockets, and many network utilities, the complete package was adopted immediately. Furthermore, with TCP/IP, it was easy for the LANs to connect to the ARPANET, and many did.

During the 1980s, additional networks, especially LANs, were connected to the ARPANET. As the scale increased, finding hosts became increasingly expensive, so **DNS** (**Domain Name System**) was created to organize machines into domains and map host names onto IP addresses. Since then, DNS has become a generalized, distributed database system for storing a variety of information related to naming. We will study it in detail in Chap. 7.

NSFNET

By the late 1970s, NSF (the U.S. National Science Foundation) saw the enormous impact the ARPANET was having on university research, allowing scientists across the country to share data and collaborate on research projects. However, to get on the ARPANET a university had to have a research contract with the DoD. Many did not have a contract. NSF's initial response was to fund the Computer Science Network (**CSNET**) in 1981. It connected computer science departments and industrial research labs to the ARPANET via dial-up and leased lines. In the late 1980s, the NSF went further and decided to design a successor to the ARPANET that would be open to all university research groups.

To have something concrete to start with, NSF decided to build a backbone network to connect its six supercomputer centers, in San Diego, Boulder, Champaign, Pittsburgh, Ithaca, and Princeton. Each supercomputer was given a little brother, consisting of an LSI-11 microcomputer called a **fuzzball**. The fuzzballs were connected with 56-kbps leased lines and formed the subnet, the same hardware technology the ARPANET used. The software technology was different however: the fuzzballs spoke TCP/IP right from the start, making it the first TCP/IP WAN.

NSF also funded some (eventually about 20) regional networks that connected to the backbone to allow users at thousands of universities, research labs, libraries, and museums to access any of the supercomputers and to communicate with one another. The complete network, including backbone and the regional networks, was called **NSFNET**. It connected to the ARPANET through a link between an

IMP and a fuzzball in the Carnegie-Mellon machine room. The first NSFNET backbone is illustrated in Fig. 1-28 superimposed on a map of the U.S.

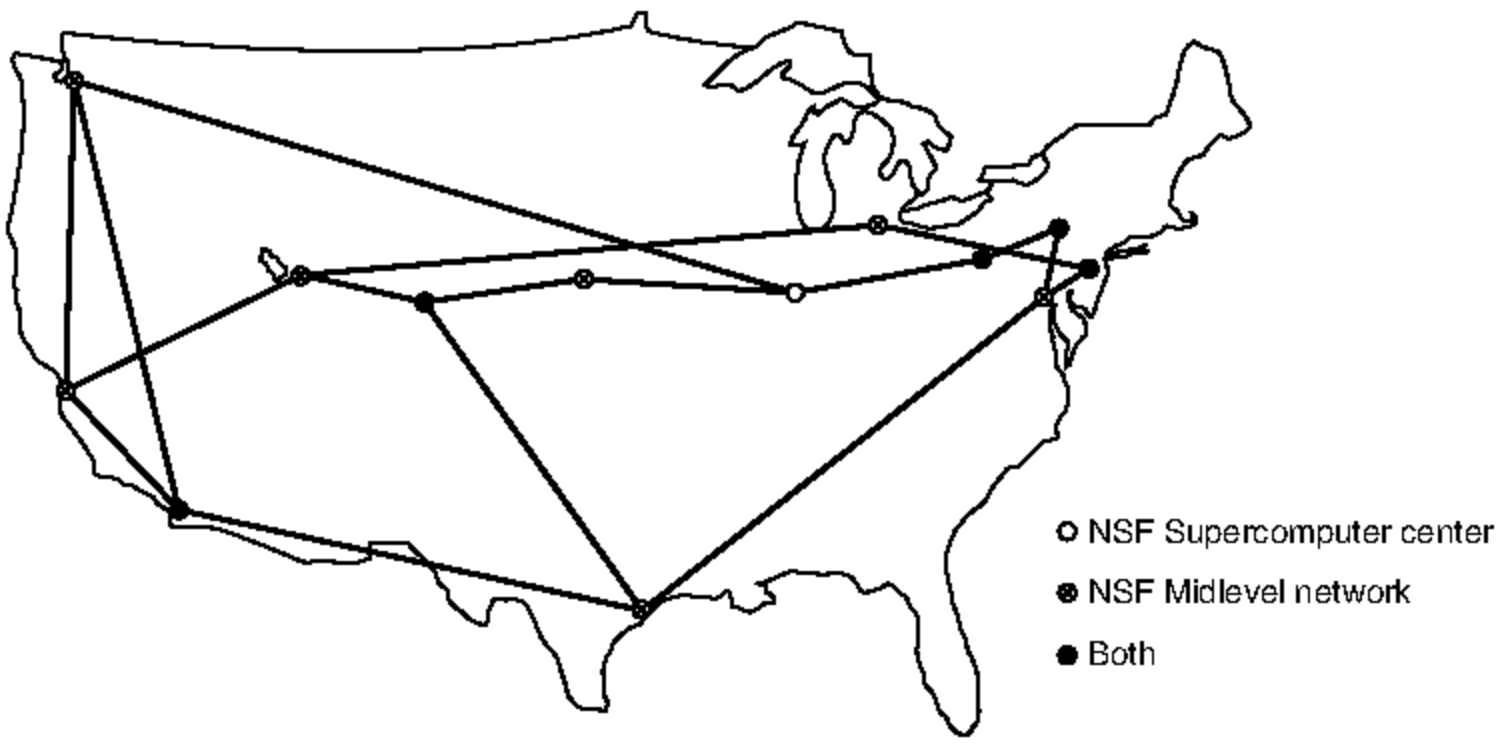

Figure 1-28. The NSFNET backbone in 1988.

NSFNET was an instantaneous success and was overloaded from the word go. NSF immediately began planning its successor and awarded a contract to the Michigan-based MERIT consortium to run it. Fiber optic channels at 448 kbps were leased from MCI (since merged with WorldCom) to provide the version 2 backbone. IBM PC-RTs were used as routers. This, too, was soon overwhelmed, and by 1990, the second backbone was upgraded to 1.5 Mbps.

As growth continued, NSF realized that the government could not continue financing networking forever. Furthermore, commercial organizations wanted to join but were forbidden by NSF's charter from using networks NSF paid for. Consequently, NSF encouraged MERIT, MCI, and IBM to form a nonprofit corporation, **ANS** (**Advanced Networks and Services**), as the first step along the road to commercialization. In 1990, ANS took over NSFNET and upgraded the 1.5-Mbps links to 45 Mbps to form **ANSNET**. This network operated for 5 years and was then sold to America Online. But by then, various companies were offering commercial IP service and it was clear the government should now get out of the networking business.

To ease the transition and make sure every regional network could communicate with every other regional network, NSF awarded contracts to four different network operators to establish a **NAP** (**Network Access Point**). These operators were PacBell (San Francisco), Ameritech (Chicago), MFS (Washington, D.C.), and Sprint (New York City, where for NAP purposes, Pennsauken, New Jersey counts as New York City). Every network operator that wanted to provide backbone service to the NSF regional networks had to connect to all the NAPs.

This arrangement meant that a packet originating on any regional network had a choice of backbone carriers to get from its NAP to the destination's NAP. Consequently, the backbone carriers were forced to compete for the regional networks' business on the basis of service and price, which was the idea, of course. As a result, the concept of a single default backbone was replaced by a commercially driven competitive infrastructure. Many people like to criticize the Federal Government for not being innovative, but in the area of networking, it was DoD and NSF that created the infrastructure that formed the basis for the Internet and then handed it over to industry to operate.

During the 1990s, many other countries and regions also built national research networks, often patterned on the ARPANET and NSFNET. These included EuropaNET and EBONE in Europe, which started out with 2-Mbps lines and then upgraded to 34-Mbps lines. Eventually, the network infrastructure in Europe was handed over to industry as well.

The Internet has changed a great deal since those early days. It exploded in size with the emergence of the World Wide Web (WWW) in the early 1990s. Recent data from the Internet Systems Consortium puts the number of visible Internet hosts at over 600 million. This guess is only a low-ball estimate, but it far exceeds the few million hosts that were around when the first conference on the WWW was held at CERN in 1994.

The way we use the Internet has also changed radically. Initially, applications such as email-for-academics, newsgroups, remote login, and file transfer dominated. Later it switched to email-for-everyman, then the Web and peer-to-peer content distribution, such as the now-shuttered Napster. Now real-time media distribution, social networks (e.g., Facebook), and microblogging (e.g., Twitter) are taking off. These switches brought richer kinds of media to the Internet and hence much more traffic. In fact, the dominant traffic on the Internet seems to change with some regularity as, for example, new and better ways to work with music or movies can become very popular very quickly.

Architecture of the Internet

The architecture of the Internet has also changed a great deal as it has grown explosively. In this section, we will attempt to give a brief overview of what it looks like today. The picture is complicated by continuous upheavals in the businesses of telephone companies (telcos), cable companies and ISPs that often make it hard to tell who is doing what. One driver of these upheavals is telecommunications convergence, in which one network is used for previously different uses. For example, in a "triple play" one company sells you telephony, TV, and Internet service over the same network connection on the assumption that this will save you money. Consequently, the description given here will be of necessity somewhat simpler than reality. And what is true today may not be true tomorrow.

The big picture is shown in Fig. 1-29. Let us examine this figure piece by piece, starting with a computer at home (at the edges of the figure). To join the Internet, the computer is connected to an **Internet Service Provider**, or simply **ISP**, from who the user purchases **Internet access** or **connectivity**. This lets the computer exchange packets with all of the other accessible hosts on the Internet. The user might send packets to surf the Web or for any of a thousand other uses, it does not matter. There are many kinds of Internet access, and they are usually distinguished by how much bandwidth they provide and how much they cost, but the most important attribute is connectivity.

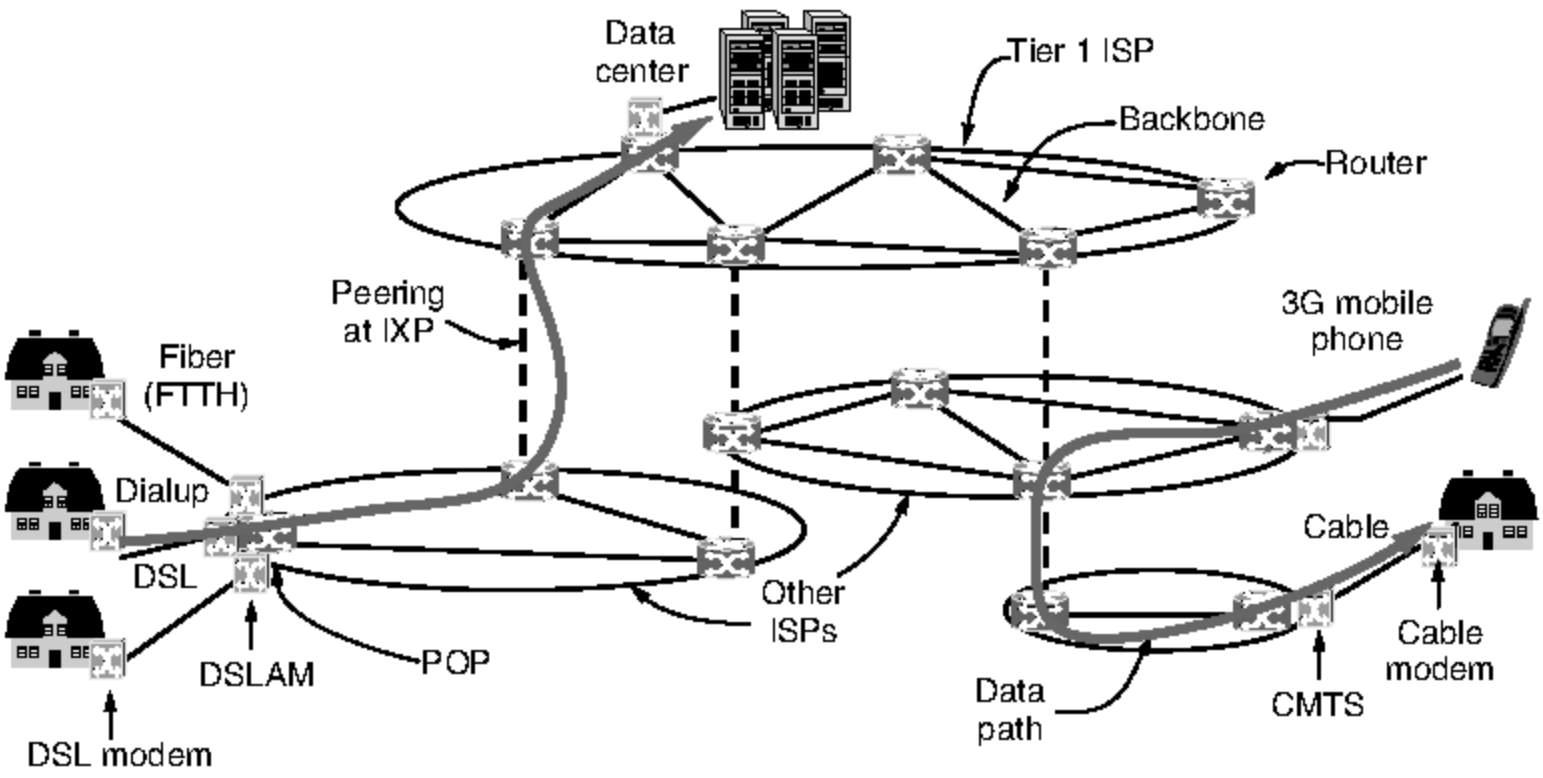

Figure 1-29. Overview of the Internet architecture.

A common way to connect to an ISP is to use the phone line to your house, in which case your phone company is your ISP. **DSL**, short for **Digital Subscriber Line**, reuses the telephone line that connects to your house for digital data transmission. The computer is connected to a device called a **DSL modem** that converts between digital packets and analog signals that can pass unhindered over the telephone line. At the other end, a device called a **DSLAM** (**Digital Subscriber Line Access Multiplexer**) converts between signals and packets.

Several other popular ways to connect to an ISP are shown in Fig. 1-29. DSL is a higher-bandwidth way to use the local telephone line than to send bits over a traditional telephone call instead of a voice conversation. That is called **dial-up** and done with a different kind of modem at both ends. The word **modem** is short for "*mo*dulator *dem*odulator" and refers to any device that converts between digital bits and analog signals.

Another method is to send signals over the cable TV system. Like DSL, this is a way to reuse existing infrastructure, in this case otherwise unused cable TV

channels. The device at the home end is called a **cable modem** and the device at the **cable headend** is called the **CMTS** (**Cable Modem Termination System**).

DSL and cable provide Internet access at rates from a small fraction of a megabit/sec to multiple megabit/sec, depending on the system. These rates are much greater than dial-up rates, which are limited to 56 kbps because of the narrow bandwidth used for voice calls. Internet access at much greater than dial-up speeds is called **broadband**. The name refers to the broader bandwidth that is used for faster networks, rather than any particular speed.

The access methods mentioned so far are limited by the bandwidth of the "last mile" or last leg of transmission. By running optical fiber to residences, faster Internet access can be provided at rates on the order of 10 to 100 Mbps. This design is called **FTTH** (**Fiber to the Home**). For businesses in commercial areas, it may make sense to lease a high-speed transmission line from the offices to the nearest ISP. For example, in North America, a T3 line runs at roughly 45 Mbps.

Wireless is used for Internet access too. An example we will explore shortly is that of 3G mobile phone networks. They can provide data delivery at rates of 1 Mbps or higher to mobile phones and fixed subscribers in the coverage area.

We can now move packets between the home and the ISP. We call the location at which customer packets enter the ISP network for service the ISP's **POP** (**Point of Presence**). We will next explain how packets are moved between the POPs of different ISPs. From this point on, the system is fully digital and packet switched.

ISP networks may be regional, national, or international in scope. We have already seen that their architecture is made up of long-distance transmission lines that interconnect routers at POPs in the different cities that the ISPs serve. This equipment is called the **backbone** of the ISP. If a packet is destined for a host served directly by the ISP, that packet is routed over the backbone and delivered to the host. Otherwise, it must be handed over to another ISP.

ISPs connect their networks to exchange traffic at **IXPs** (**Internet eXchange Points**). The connected ISPs are said to **peer** with each other. There are many IXPs in cities around the world. They are drawn vertically in Fig. 1-29 because ISP networks overlap geographically. Basically, an IXP is a room full of routers, at least one per ISP. A LAN in the room connects all the routers, so packets can be forwarded from any ISP backbone to any other ISP backbone. IXPs can be large and independently owned facilities. One of the largest is the Amsterdam Internet Exchange, to which hundreds of ISPs connect and through which they exchange hundreds of gigabits/sec of traffic.

The peering that happens at IXPs depends on the business relationships between ISPs. There are many possible relationships. For example, a small ISP might pay a larger ISP for Internet connectivity to reach distant hosts, much as a customer purchases service from an Internet provider. In this case, the small ISP is said to pay for **transit**. Alternatively, two large ISPs might decide to exchange

traffic so that each ISP can deliver some traffic to the other ISP without having to pay for transit. One of the many paradoxes of the Internet is that ISPs who publicly compete with one another for customers often privately cooperate to do peering (Metz, 2001).

The path a packet takes through the Internet depends on the peering choices of the ISPs. If the ISP delivering a packet peers with the destination ISP, it might deliver the packet directly to its peer. Otherwise, it might route the packet to the nearest place at which it connects to a paid transit provider so that provider can deliver the packet. Two example paths across ISPs are drawn in Fig. 1-29. Often, the path a packet takes will not be the shortest path through the Internet.

At the top of the food chain are a small handful of companies, like AT&T and Sprint, that operate large international backbone networks with thousands of routers connected by high-bandwidth fiber optic links. These ISPs do not pay for transit. They are usually called **tier 1** ISPs and are said to form the backbone of the Internet, since everyone else must connect to them to be able to reach the entire Internet.

Companies that provide lots of content, such as Google and Yahoo!, locate their computers in **data centers** that are well connected to the rest of the Internet. These data centers are designed for computers, not humans, and may be filled with rack upon rack of machines called a **server farm**. **Colocation** or **hosting** data centers let customers put equipment such as servers at ISP POPs so that short, fast connections can be made between the servers and the ISP backbones. The Internet hosting industry has become increasingly virtualized so that it is now common to rent a virtual machine that is run on a server farm instead of installing a physical computer. These data centers are so large (tens or hundreds of thousands of machines) that electricity is a major cost, so data centers are sometimes built in areas where electricity is cheap.

This ends our quick tour of the Internet. We will have a great deal to say about the individual components and their design, algorithms, and protocols in subsequent chapters. One further point worth mentioning here is that what it means to be on the Internet is changing. It used to be that a machine was on the Internet if it: (1) ran the TCP/IP protocol stack; (2) had an IP address; and (3) could send IP packets to all the other machines on the Internet. However, ISPs often reuse IP addresses depending on which computers are in use at the moment, and home networks often share one IP address between multiple computers. This practice undermines the second condition. Security measures such as firewalls can also partly block computers from receiving packets, undermining the third condition. Despite these difficulties, it makes sense to regard such machines as being on the Internet while they are connected to their ISPs.

Also worth mentioning in passing is that some companies have interconnected all their existing internal networks, often using the same technology as the Internet. These **intranets** are typically accessible only on company premises or from company notebooks but otherwise work the same way as the Internet.

1.5.2 Third-Generation Mobile Phone Networks

People love to talk on the phone even more than they like to surf the Internet, and this has made the mobile phone network the most successful network in the world. It has more than four billion subscribers worldwide. To put this number in perspective, it is roughly 60% of the world's population and more than the number of Internet hosts and fixed telephone lines combined (ITU, 2009).

The architecture of the mobile phone network has changed greatly over the past 40 years along with its tremendous growth. First-generation mobile phone systems transmitted voice calls as continuously varying (analog) signals rather than sequences of (digital) bits. **AMPS** (**Advanced Mobile Phone System**), which was deployed in the United States in 1982, was a widely used first-generation system. Second-generation mobile phone systems switched to transmitting voice calls in digital form to increase capacity, improve security, and offer text messaging. **GSM** (**Global System for Mobile communications**), which was deployed starting in 1991 and has become the most widely used mobile phone system in the world, is a 2G system.

The third generation, or 3G, systems were initially deployed in 2001 and offer both digital voice and broadband digital data services. They also come with a lot of jargon and many different standards to choose from. 3G is loosely defined by the ITU (an international standards body we will discuss in the next section) as providing rates of at least 2 Mbps for stationary or walking users and 384 kbps in a moving vehicle. **UMTS** (**Universal Mobile Telecommunications System**), also called **WCDMA** (**Wideband Code Division Multiple Access**), is the main 3G system that is being rapidly deployed worldwide. It can provide up to 14 Mbps on the downlink and almost 6 Mbps on the uplink. Future releases will use multiple antennas and radios to provide even greater speeds for users.

The scarce resource in 3G systems, as in 2G and 1G systems before them, is radio spectrum. Governments license the right to use parts of the spectrum to the mobile phone network operators, often using a spectrum auction in which network operators submit bids. Having a piece of licensed spectrum makes it easier to design and operate systems, since no one else is allowed transmit on that spectrum, but it often costs a serious amount of money. In the UK in 2000, for example, five 3G licenses were auctioned for a total of about $40 billion.

It is the scarcity of spectrum that led to the **cellular network** design shown in Fig. 1-30 that is now used for mobile phone networks. To manage the radio interference between users, the coverage area is divided into cells. Within a cell, users are assigned channels that do not interfere with each other and do not cause too much interference for adjacent cells. This allows for good reuse of the spectrum, or **frequency reuse**, in the neighboring cells, which increases the capacity of the network. In 1G systems, which carried each voice call on a specific frequency band, the frequencies were carefully chosen so that they did not conflict with neighboring cells. In this way, a given frequency might only be reused once

in several cells. Modern 3G systems allow each cell to use all frequencies, but in a way that results in a tolerable level of interference to the neighboring cells. There are variations on the cellular design, including the use of directional or sectored antennas on cell towers to further reduce interference, but the basic idea is the same.

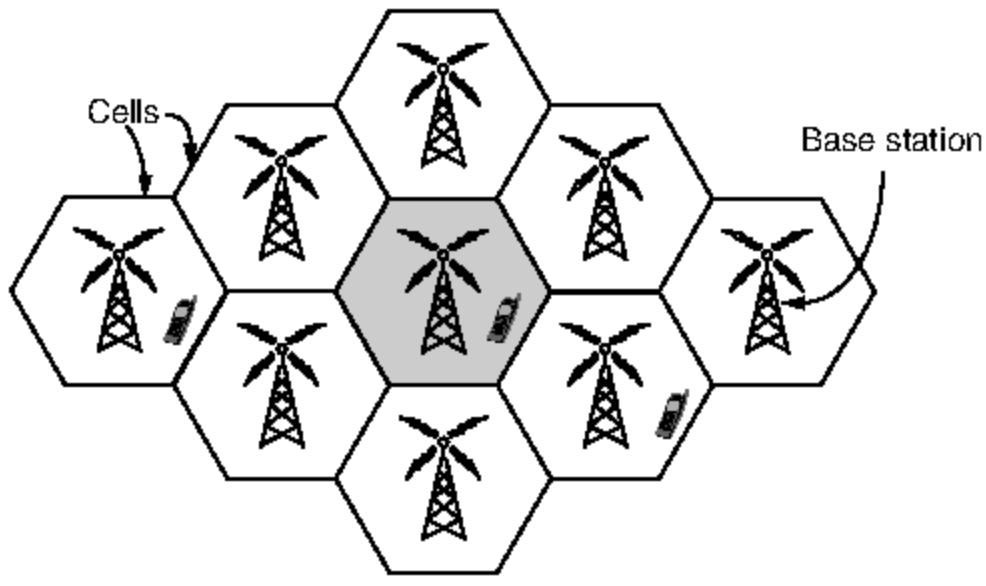

Figure 1-30. Cellular design of mobile phone networks.

The architecture of the mobile phone network is very different than that of the Internet. It has several parts, as shown in the simplified version of the UMTS architecture in Fig. 1-31. First, there is the **air interface**. This term is a fancy name for the radio communication protocol that is used over the air between the mobile device (e.g., the cell phone) and the **cellular base station**. Advances in the air interface over the past decades have greatly increased wireless data rates. The UMTS air interface is based on **Code Division Multiple Access** (**CDMA**), a technique that we will study in Chap. 2.

The cellular base station together with its controller forms the **radio access network**. This part is the wireless side of the mobile phone network. The controller node or **RNC** (**Radio Network Controller**) controls how the spectrum is used. The base station implements the air interface. It is called **Node B**, a temporary label that stuck.

The rest of the mobile phone network carries the traffic for the radio access network. It is called the **core network**. The UMTS core network evolved from the core network used for the 2G GSM system that came before it. However, something surprising is happening in the UMTS core network.

Since the beginning of networking, a war has been going on between the people who support packet networks (i.e., connectionless subnets) and the people who support circuit networks (i.e., connection-oriented subnets). The main proponents of packets come from the Internet community. In a connectionless design, every packet is routed independently of every other packet. As a consequence, if some routers go down during a session, no harm will be done as long as the system can

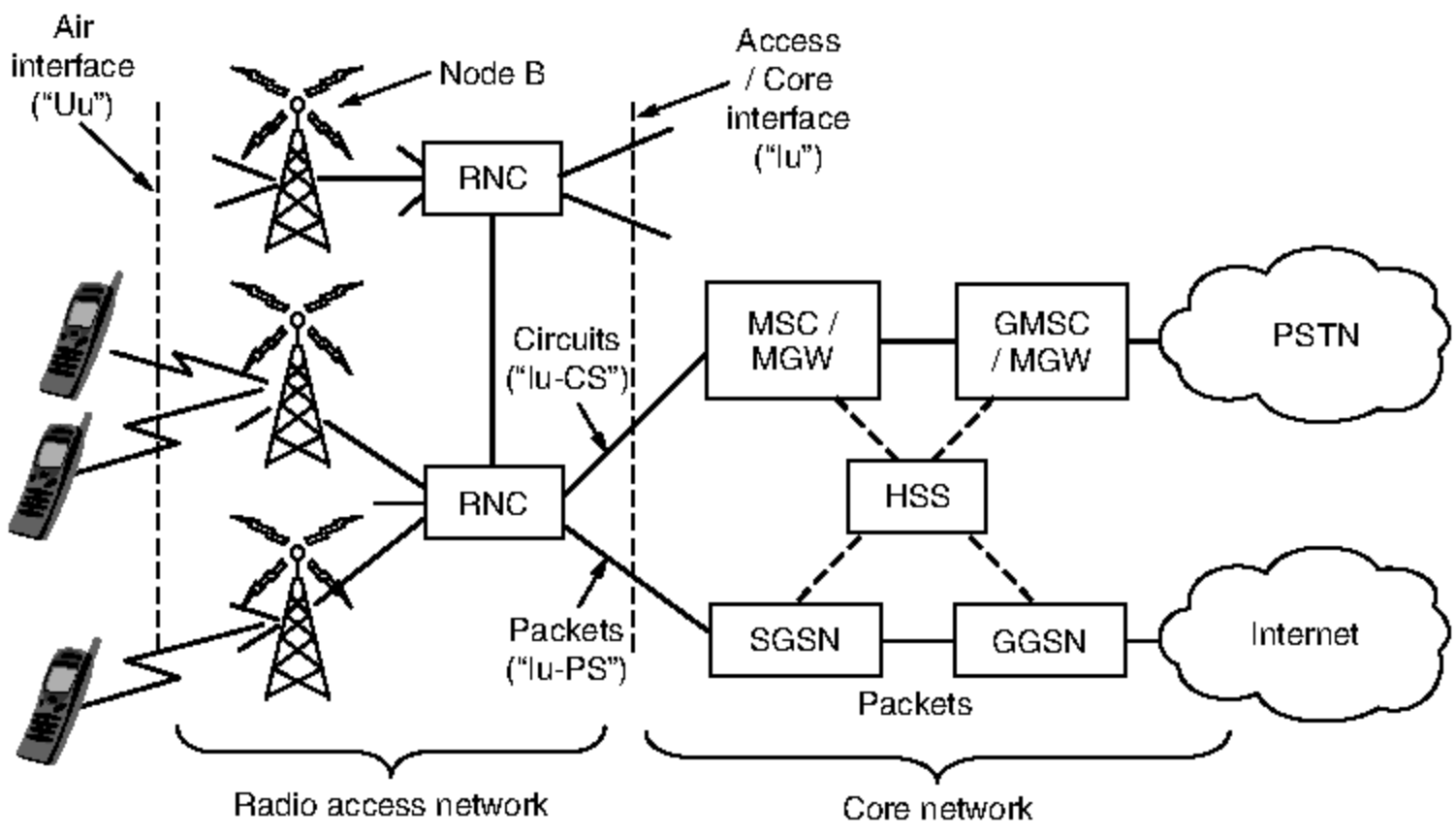

Figure 1-31. Architecture of the UMTS 3G mobile phone network.

dynamically reconfigure itself so that subsequent packets can find some route to the destination, even if it is different from that which previous packets used.

The circuit camp comes from the world of telephone companies. In the telephone system, a caller must dial the called party's number and wait for a connection before talking or sending data. This connection setup establishes a route through the telephone system that is maintained until the call is terminated. All words or packets follow the same route. If a line or switch on the path goes down, the call is aborted, making it less fault tolerant than a connectionless design.

The advantage of circuits is that they can support quality of service more easily. By setting up a connection in advance, the subnet can reserve resources such as link bandwidth, switch buffer space, and CPU. If an attempt is made to set up a call and insufficient resources are available, the call is rejected and the caller gets a kind of busy signal. In this way, once a connection has been set up, the connection will get good service.

With a connectionless network, if too many packets arrive at the same router at the same moment, the router will choke and probably lose packets. The sender will eventually notice this and resend them, but the quality of service will be jerky and unsuitable for audio or video unless the network is lightly loaded. Needless to say, providing adequate audio quality is something telephone companies care about very much, hence their preference for connections.

The surprise in Fig. 1-31 is that there is both packet and circuit switched equipment in the core network. This shows the mobile phone network in transition, with mobile phone companies able to implement one or sometimes both of

the alternatives. Older mobile phone networks used a circuit-switched core in the style of the traditional phone network to carry voice calls. This legacy is seen in the UMTS network with the **MSC** (**Mobile Switching Center**), **GMSC** (**Gateway Mobile Switching Center**), and **MGW** (**Media Gateway**) elements that set up connections over a circuit-switched core network such as the **PSTN** (**Public Switched Telephone Network**).

Data services have become a much more important part of the mobile phone network than they used to be, starting with text messaging and early packet data services such as **GPRS** (**General Packet Radio Service**) in the GSM system. These older data services ran at tens of kbps, but users wanted more. Newer mobile phone networks carry packet data at rates of multiple Mbps. For comparison, a voice call is carried at a rate of 64 kbps, typically 3–4x less with compression.

To carry all this data, the UMTS core network nodes connect directly to a packet-switched network. The **SGSN** (**Serving GPRS Support Node**) and the **GGSN** (**Gateway GPRS Support Node**) deliver data packets to and from mobiles and interface to external packet networks such as the Internet.

This transition is set to continue in the mobile phone networks that are now being planned and deployed. Internet protocols are even used on mobiles to set up connections for voice calls over a packet data network, in the manner of voice-over-IP. IP and packets are used all the way from the radio access through to the core network. Of course, the way that IP networks are designed is also changing to support better quality of service. If it did not, then problems with chopped-up audio and jerky video would not impress paying customers. We will return to this subject in Chap. 5.

Another difference between mobile phone networks and the traditional Internet is mobility. When a user moves out of the range of one cellular base station and into the range of another one, the flow of data must be re-routed from the old to the new cell base station. This technique is known as **handover** or **handoff**, and it is illustrated in Fig. 1-32.

Figure 1-32. Mobile phone handover (a) before, (b) after.

Either the mobile device or the base station may request a handover when the quality of the signal drops. In some cell networks, usually those based on CDMA

technology, it is possible to connect to the new base station before disconnecting from the old base station. This improves the connection quality for the mobile because there is no break in service; the mobile is actually connected to two base stations for a short while. This way of doing a handover is called a **soft handover** to distinguish it from a **hard handover**, in which the mobile disconnects from the old base station before connecting to the new one.

A related issue is how to find a mobile in the first place when there is an incoming call. Each mobile phone network has a **HSS** (**Home Subscriber Server**) in the core network that knows the location of each subscriber, as well as other profile information that is used for authentication and authorization. In this way, each mobile can be found by contacting the HSS.

A final area to discuss is security. Historically, phone companies have taken security much more seriously than Internet companies for a long time because of the need to bill for service and avoid (payment) fraud. Unfortunately that is not saying much. Nevertheless, in the evolution from 1G through 3G technologies, mobile phone companies have been able to roll out some basic security mechanisms for mobiles.

Starting with the 2G GSM system, the mobile phone was divided into a handset and a removable chip containing the subscriber's identity and account information. The chip is informally called a **SIM card**, short for **Subscriber Identity Module**. SIM cards can be switched to different handsets to activate them, and they provide a basis for security. When GSM customers travel to other countries on vacation or business, they often bring their handsets but buy a new SIM card for few dollars upon arrival in order to make local calls with no roaming charges.

To reduce fraud, information on SIM cards is also used by the mobile phone network to authenticate subscribers and check that they are allowed to use the network. With UMTS, the mobile also uses the information on the SIM card to check that it is talking to a legitimate network.

Another aspect of security is privacy. Wireless signals are broadcast to all nearby receivers, so to make it difficult to eavesdrop on conversations, cryptographic keys on the SIM card are used to encrypt transmissions. This approach provides much better privacy than in 1G systems, which were easily tapped, but is not a panacea due to weaknesses in the encryption schemes.

Mobile phone networks are destined to play a central role in future networks. They are now more about mobile broadband applications than voice calls, and this has major implications for the air interfaces, core network architecture, and security of future networks. 4G technologies that are faster and better are on the drawing board under the name of **LTE** (**Long Term Evolution**), even as 3G design and deployment continues. Other wireless technologies also offer broadband Internet access to fixed and mobile clients, notably 802.16 networks under the common name of **WiMAX**. It is entirely possible that LTE and WiMAX are on a collision course with each other and it is hard to predict what will happen to them.

1.5.3 Wireless LANs: 802.11

Almost as soon as laptop computers appeared, many people had a dream of walking into an office and magically having their laptop computer be connected to the Internet. Consequently, various groups began working on ways to accomplish this goal. The most practical approach is to equip both the office and the laptop computers with short-range radio transmitters and receivers to allow them to talk.

Work in this field rapidly led to wireless LANs being marketed by a variety of companies. The trouble was that no two of them were compatible. The proliferation of standards meant that a computer equipped with a brand *X* radio would not work in a room equipped with a brand *Y* base station. In the mid 1990s, the industry decided that a wireless LAN standard might be a good idea, so the IEEE committee that had standardized wired LANs was given the task of drawing up a wireless LAN standard.

The first decision was the easiest: what to call it. All the other LAN standards had numbers like 802.1, 802.2, and 802.3, up to 802.10, so the wireless LAN standard was dubbed 802.11. A common slang name for it is **WiFi** but it is an important standard and deserves respect, so we will call it by its proper name, 802.11.

The rest was harder. The first problem was to find a suitable frequency band that was available, preferably worldwide. The approach taken was the opposite of that used in mobile phone networks. Instead of expensive, licensed spectrum, 802.11 systems operate in unlicensed bands such as the **ISM** (**Industrial, Scientific, and Medical**) bands defined by ITU-R (e.g., 902-928 MHz, 2.4-2.5 GHz, 5.725-5.825 GHz). All devices are allowed to use this spectrum provided that they limit their transmit power to let different devices coexist. Of course, this means that 802.11 radios may find themselves competing with cordless phones, garage door openers, and microwave ovens.

802.11 networks are made up of clients, such as laptops and mobile phones, and infrastructure called **APs** (**access points**) that is installed in buildings. Access points are sometimes called **base stations**. The access points connect to the wired network, and all communication between clients goes through an access point. It is also possible for clients that are in radio range to talk directly, such as two computers in an office without an access point. This arrangement is called an **ad hoc network**. It is used much less often than the access point mode. Both modes are shown in Fig. 1-33.

802.11 transmission is complicated by wireless conditions that vary with even small changes in the environment. At the frequencies used for 802.11, radio signals can be reflected off solid objects so that multiple echoes of a transmission may reach a receiver along different paths. The echoes can cancel or reinforce each other, causing the received signal to fluctuate greatly. This phenomenon is called **multipath fading**, and it is shown in Fig. 1-34.

The key idea for overcoming variable wireless conditions is **path diversity**, or the sending of information along multiple, independent paths. In this way, the

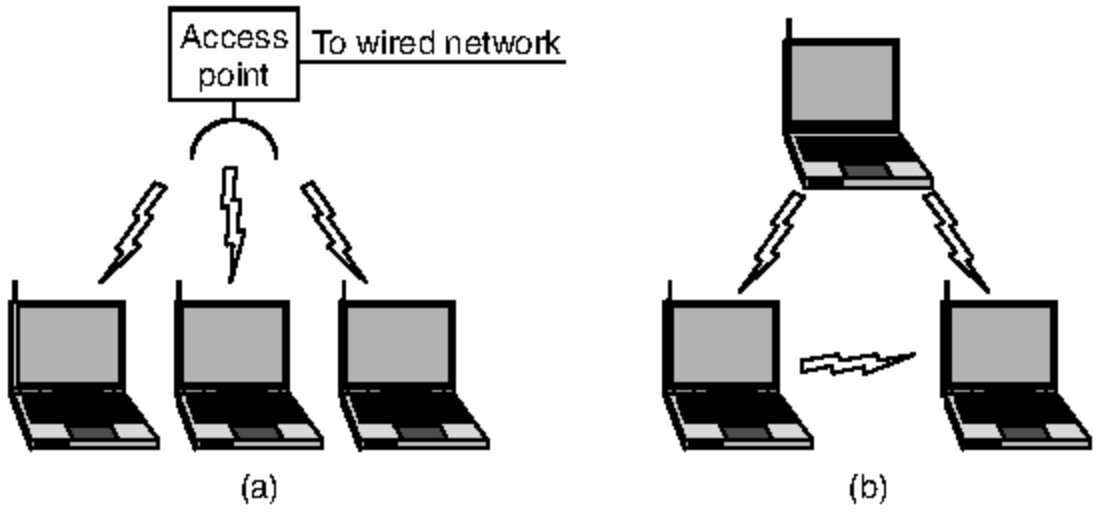

Figure 1-33. (a) Wireless network with an access point. (b) Ad hoc network.

information is likely to be received even if one of the paths happens to be poor due to a fade. These independent paths are typically built into the digital modulation scheme at the physical layer. Options include using different frequencies across the allowed band, following different spatial paths between different pairs of antennas, or repeating bits over different periods of time.

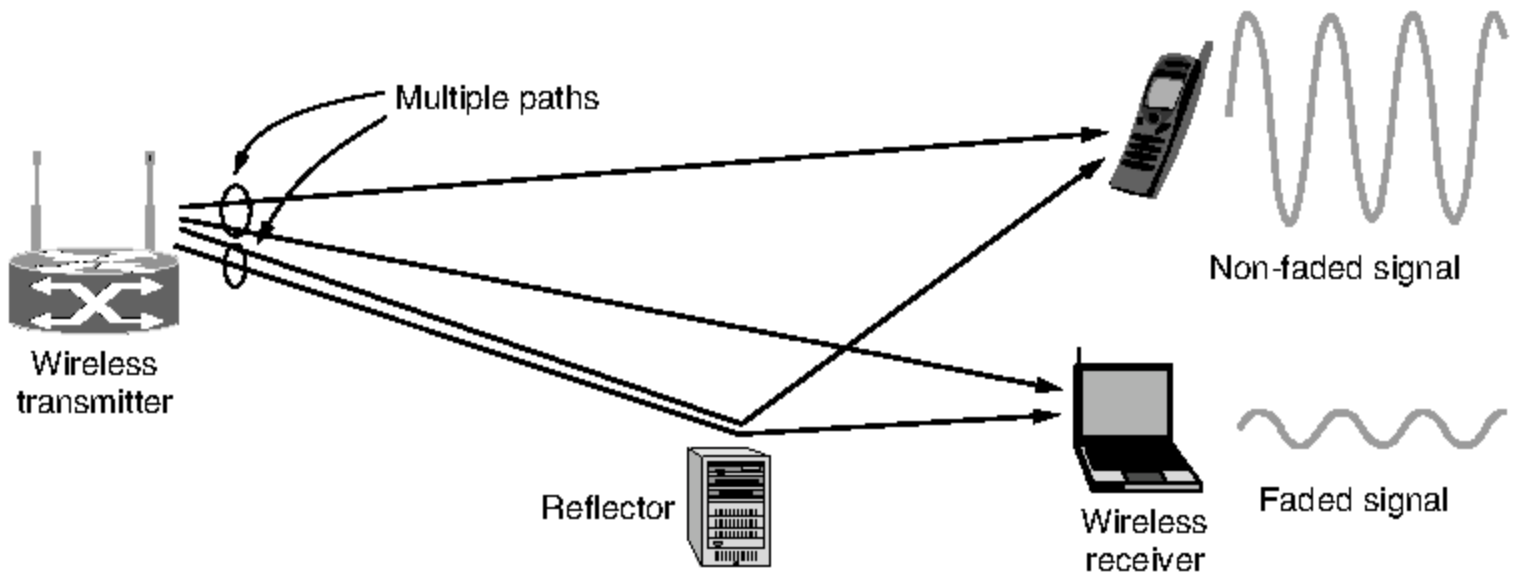

Figure 1-34. Multipath fading.

Different versions of 802.11 have used all of these techniques. The initial (1997) standard defined a wireless LAN that ran at either 1 Mbps or 2 Mbps by hopping between frequencies or spreading the signal across the allowed spectrum. Almost immediately, people complained that it was too slow, so work began on faster standards. The spread spectrum design was extended and became the (1999) 802.11b standard running at rates up to 11 Mbps. The 802.11a (1999) and 802.11g (2003) standards switched to a different modulation scheme called **OFDM** (**Orthogonal Frequency Division Multiplexing**). It divides a wide band of spectrum into many narrow slices over which different bits are sent in parallel. This improved scheme, which we will study in Chap. 2, boosted the 802.11a/g bit

rates up to 54 Mbps. That is a significant increase, but people still wanted more throughput to support more demanding uses. The latest version is 802.11n (2009). It uses wider frequency bands and up to four antennas per computer to achieve rates up to 450 Mbps.

Since wireless is inherently a broadcast medium, 802.11 radios also have to deal with the problem that multiple transmissions that are sent at the same time will collide, which may interfere with reception. To handle this problem, 802.11 uses a **CSMA** (**Carrier Sense Multiple Access**) scheme that draws on ideas from classic wired Ethernet, which, ironically, drew from an early wireless network developed in Hawaii and called **ALOHA**. Computers wait for a short random interval before transmitting, and defer their transmissions if they hear that someone else is already transmitting. This scheme makes it less likely that two computers will send at the same time. It does not work as well as in the case of wired networks, though. To see why, examine Fig. 1-35. Suppose that computer *A* is transmitting to computer *B*, but the radio range of *A*'s transmitter is too short to reach computer *C*. If *C* wants to transmit to *B* it can listen before starting, but the fact that it does not hear anything does not mean that its transmission will succeed. The inability of *C* to hear *A* before starting causes some collisions to occur. After any collision, the sender then waits another, longer, random delay and retransmits the packet. Despite this and some other issues, the scheme works well enough in practice.

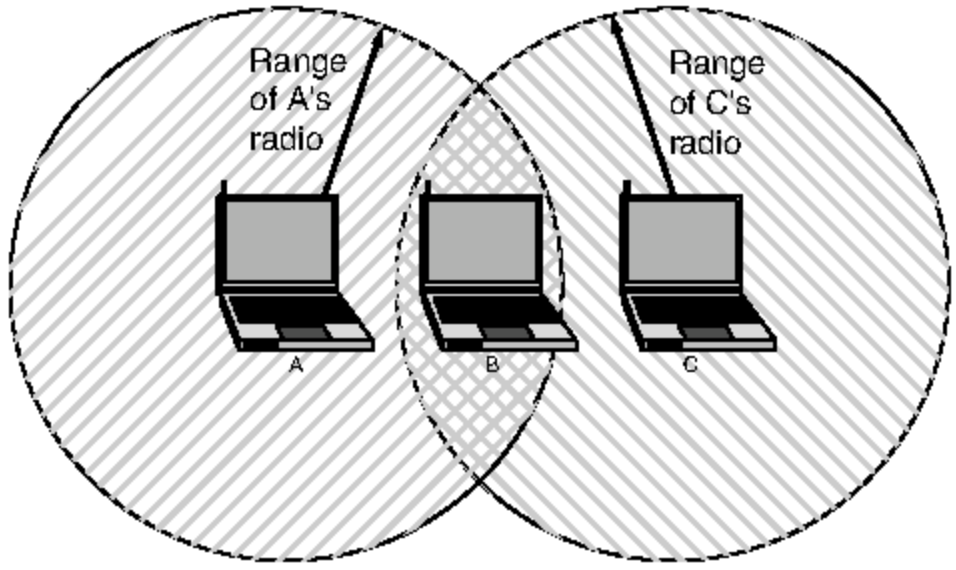

Figure 1-35. The range of a single radio may not cover the entire system.

Another problem is that of mobility. If a mobile client is moved away from the access point it is using and into the range of a different access point, some way of handing it off is needed. The solution is that an 802.11 network can consist of multiple cells, each with its own access point, and a distribution system that connects the cells. The distribution system is often switched Ethernet, but it can use any technology. As the clients move, they may find another access point with a better signal than the one they are currently using and change their association. From the outside, the entire system looks like a single wired LAN.

That said, mobility in 802.11 has been of limited value so far compared to mobility in the mobile phone network. Typically, 802.11 is used by nomadic clients that go from one fixed location to another, rather than being used on-the-go. Mobility is not really needed for nomadic usage. Even when 802.11 mobility is used, it extends over a single 802.11 network, which might cover at most a large building. Future schemes will need to provide mobility across different networks and across different technologies (e.g., 802.21).

Finally, there is the problem of security. Since wireless transmissions are broadcast, it is easy for nearby computers to receive packets of information that were not intended for them. To prevent this, the 802.11 standard included an encryption scheme known as **WEP** (**Wired Equivalent Privacy**). The idea was to make wireless security like that of wired security. It is a good idea, but unfortunately the scheme was flawed and soon broken (Borisov et al., 2001). It has since been replaced with newer schemes that have different cryptographic details in the 802.11i standard, also called **WiFi Protected Access**, initially called **WPA** but now replaced by **WPA2**.

802.11 has caused a revolution in wireless networking that is set to continue. Beyond buildings, it is starting to be installed in trains, planes, boats, and automobiles so that people can surf the Internet wherever they go. Mobile phones and all manner of consumer electronics, from game consoles to digital cameras, can communicate with it. We will come back to it in detail in Chap. 4.

1.5.4 RFID and Sensor Networks

The networks we have studied so far are made up of computing devices that are easy to recognize, from computers to mobile phones. With **Radio Frequency IDentification** (**RFID**), everyday objects can also be part of a computer network.

An RFID tag looks like a postage stamp-sized sticker that can be affixed to (or embedded in) an object so that it can be tracked. The object might be a cow, a passport, a book or a shipping pallet. The tag consists of a small microchip with a unique identifier and an antenna that receives radio transmissions. RFID readers installed at tracking points find tags when they come into range and interrogate them for their information as shown in Fig. 1-36. Applications include checking identities, managing the supply chain, timing races, and replacing barcodes.

There are many kinds of RFID, each with different properties, but perhaps the most fascinating aspect of RFID technology is that most RFID tags have neither an electric plug nor a battery. Instead, all of the energy needed to operate them is supplied in the form of radio waves by RFID readers. This technology is called **passive RFID** to distinguish it from the (less common) **active RFID** in which there is a power source on the tag.

One common form of RFID is **UHF RFID** (**Ultra-High Frequency RFID**). It is used on shipping pallets and some drivers licenses. Readers send signals in

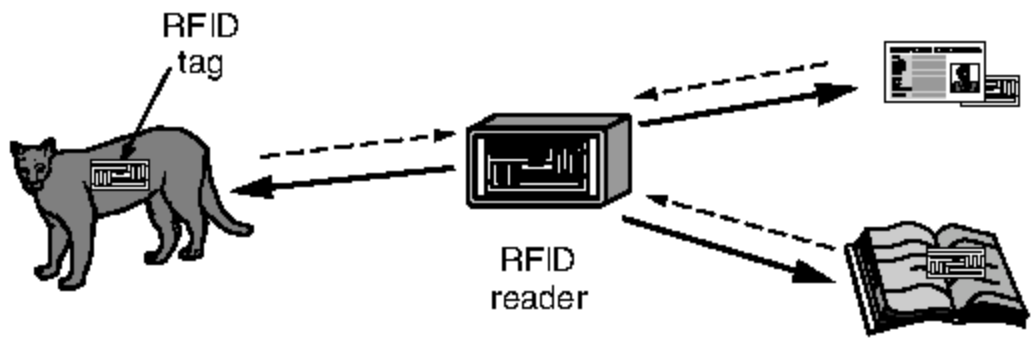

Figure 1-36. RFID used to network everyday objects.

the 902-928 MHz band in the United States. Tags communicate at distances of several meters by changing the way they reflect the reader signals; the reader is able to pick up these reflections. This way of operating is called **backscatter**.

Another popular kind of RFID is **HF RFID** (**High Frequency RFID**). It operates at 13.56 MHz and is likely to be in your passport, credit cards, books, and noncontact payment systems. HF RFID has a short range, typically a meter or less, because the physical mechanism is based on induction rather than backscatter. There are also other forms of RFID using other frequencies, such as **LF RFID** (**Low Frequency RFID**), which was developed before HF RFID and used for animal tracking. It is the kind of RFID likely to be in your cat.

RFID readers must somehow solve the problem of dealing with multiple tags within reading range. This means that a tag cannot simply respond when it hears a reader, or the signals from multiple tags may collide. The solution is similar to the approach taken in 802.11: tags wait for a short random interval before responding with their identification, which allows the reader to narrow down individual tags and interrogate them further.

Security is another problem. The ability of RFID readers to easily track an object, and hence the person who uses it, can be an invasion of privacy. Unfortunately, it is difficult to secure RFID tags because they lack the computation and communication power to run strong cryptographic algorithms. Instead, weak measures like passwords (which can easily be cracked) are used. If an identity card can be remotely read by an official at a border, what is to stop the same card from being tracked by other people without your knowledge? Not much.

RFID tags started as identification chips, but are rapidly turning into full-fledged computers. For example, many tags have memory that can be updated and later queried, so that information about what has happened to the tagged object can be stored with it. Rieback et al. (2006) demonstrated that this means that all of the usual problems of computer malware apply, only now your cat or your passport might be used to spread an RFID virus.

A step up in capability from RFID is the **sensor network**. Sensor networks are deployed to monitor aspects of the physical world. So far, they have mostly been used for scientific experimentation, such as monitoring bird habitats, volcanic activity, and zebra migration, but business applications including healthcare,

monitoring equipment for vibration, and tracking of frozen, refrigerated, or otherwise perishable goods cannot be too far behind.

Sensor nodes are small computers, often the size of a key fob, that have temperature, vibration, and other sensors. Many nodes are placed in the environment that is to be monitored. Typically, they have batteries, though they may scavenge energy from vibrations or the sun. As with RFID, having enough energy is a key challenge, and the nodes must communicate carefully to be able to deliver their sensor information to an external collection point. A common strategy is for the nodes to self-organize to relay messages for each other, as shown in Fig. 1-37. This design is called a **multihop network**.

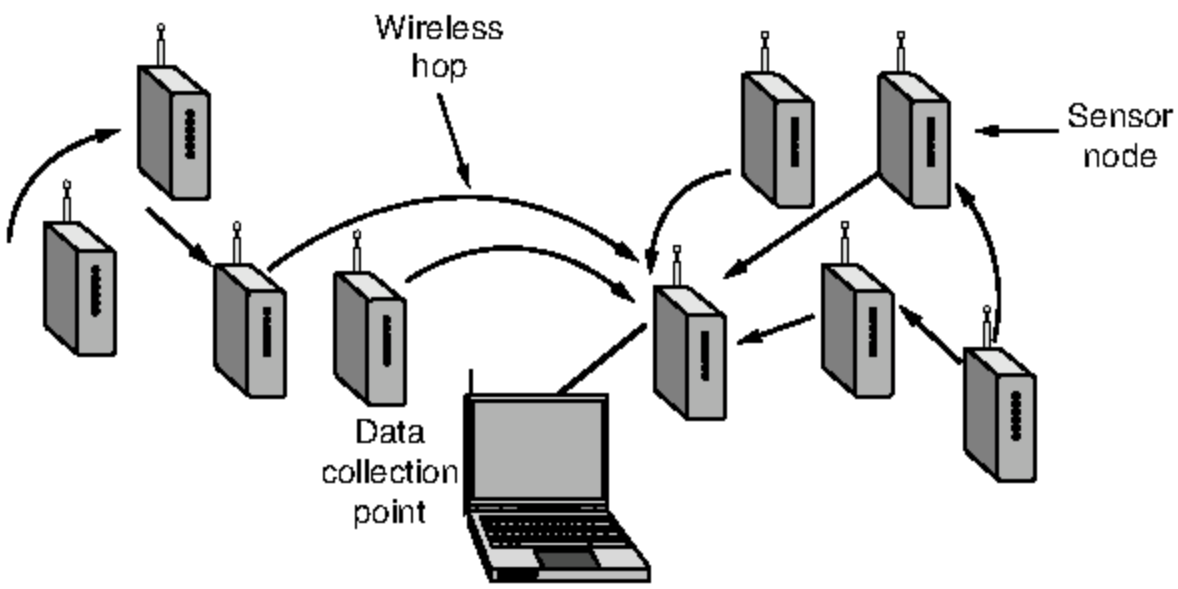

Figure 1-37. Multihop topology of a sensor network.

RFID and sensor networks are likely to become much more capable and pervasive in the future. Researchers have already combined the best of both technologies by prototyping programmable RFID tags with light, movement, and other sensors (Sample et al., 2008).

1.6 NETWORK STANDARDIZATION

Many network vendors and suppliers exist, each with its own ideas of how things should be done. Without coordination, there would be complete chaos, and users would get nothing done. The only way out is to agree on some network standards. Not only do good standards allow different computers to communicate, but they also increase the market for products adhering to the standards. A larger market leads to mass production, economies of scale in manufacturing, better implementations, and other benefits that decrease price and further increase acceptance.

In this section we will take a quick look at the important but little-known, world of international standardization. But let us first discuss what belongs in a

standard. A reasonable person might assume that a standard tells you how a protocol should work so that you can do a good job of implementing it. That person would be wrong.

Standards define what is needed for interoperability: no more, no less. That lets the larger market emerge and also lets companies compete on the basis of how good their products are. For example, the 802.11 standard defines many transmission rates but does not say when a sender should use which rate, which is a key factor in good performance. That is up to whoever makes the product. Often getting to interoperability this way is difficult, since there are many implementation choices and standards usually define many options. For 802.11, there were so many problems that, in a strategy that has become common practice, a trade group called the **WiFi Alliance** was started to work on interoperability within the 802.11 standard.

Similarly, a protocol standard defines the protocol over the wire but not the service interface inside the box, except to help explain the protocol. Real service interfaces are often proprietary. For example, the way TCP interfaces to IP within a computer does not matter for talking to a remote host. It only matters that the remote host speaks TCP/IP. In fact, TCP and IP are commonly implemented together without any distinct interface. That said, good service interfaces, like good APIs, are valuable for getting protocols used, and the best ones (such as Berkeley sockets) can become very popular.

Standards fall into two categories: de facto and de jure. **De facto** (Latin for "from the fact") standards are those that have just happened, without any formal plan. HTTP, the protocol on which the Web runs, started life as a de facto standard. It was part of early WWW browsers developed by Tim Berners-Lee at CERN, and its use took off with the growth of the Web. Bluetooth is another example. It was originally developed by Ericsson but now everyone is using it.

De jure (Latin for "by law") standards, in contrast, are adopted through the rules of some formal standardization body. International standardization authorities are generally divided into two classes: those established by treaty among national governments, and those comprising voluntary, nontreaty organizations. In the area of computer network standards, there are several organizations of each type, notably ITU, ISO, IETF and IEEE, all of which we will discuss below.

In practice, the relationships between standards, companies, and standardization bodies are complicated. De facto standards often evolve into de jure standards, especially if they are successful. This happened in the case of HTTP, which was quickly picked up by IETF. Standards bodies often ratify each others' standards, in what looks like patting one another on the back, to increase the market for a technology. These days, many ad hoc business alliances that are formed around particular technologies also play a significant role in developing and refining network standards. For example, **3GPP** (**Third Generation Partnership Project**) is a collaboration between telecommunications associations that drives the UMTS 3G mobile phone standards.