Chapter 7

internetworking

iMac, iPod, iTunes, iPhone, iNternet. Right?

In 1998, Apple Computer had been losing market share for several years and many were predicting the company's imminent demise. Then, Apple introduced a new desktop computer named the iMac. Since then, the iMac and everything else the company has named iSomething has sparked a remarkable recovery.

The "i" in iMac stands for Internet. For some unknown reason, Apple decided to replace the capital "I" that belongs at the beginning of the name of the network with a lower-case "i". This change is interesting because within the field of networking "Internet" and "internet" have different meanings. "Internet" refers to the very important network most of us use every day. On the other hand, "internet" refers to a technical approach to building a network that happens to be essential to the success of THE Internet. Basically, "internet" is a concept, "Internet" is an example.

The first step in understanding the internet concept is to think about the words "internet", "interstellar", "interstate", "interpersonal", "international", "interdisciplinary", and "interfaith" (and even "interrupt"). As a prefix, "inter" means "between" or "among". Up to this point, our discussion of networks has been focused on communications between computers, but the network we all use is not called the Intercomputer. This is because the Internet is really a network of networks rather than a network of computers. To understand internets, including the Internet, our focus must shift from communications between computers to communications between networks.

In the preceding chapters, we have presented several distinct ways to construct networks including Ethernets, wireless networks, and switched networks. There are many variations on these basic approaches and many other approaches we have not even considered. The consequence is that there will never be one network connecting all of the computers in the world. As I sit typing these words, my computer is enjoying the free wireless network provided by a local coffee shop. At home, it connects through a DSL connection. Yet in both of these locations, I expect to be able to use my computer to communicate with servers connected to the Ethernet maintained by the computer science department where I work. To make this possible, we must not only provide communications between computers. We need communications between networks.

In this chapter, we will explore how the concept of an internet can provide an elegant way to support such communications between networks. We will begin by identifying the fundamental concepts behind the notion of an internet. We will then move on to consider the concrete details of the implementation of these concepts within the Internet. If we succeed, by the end of this chapter

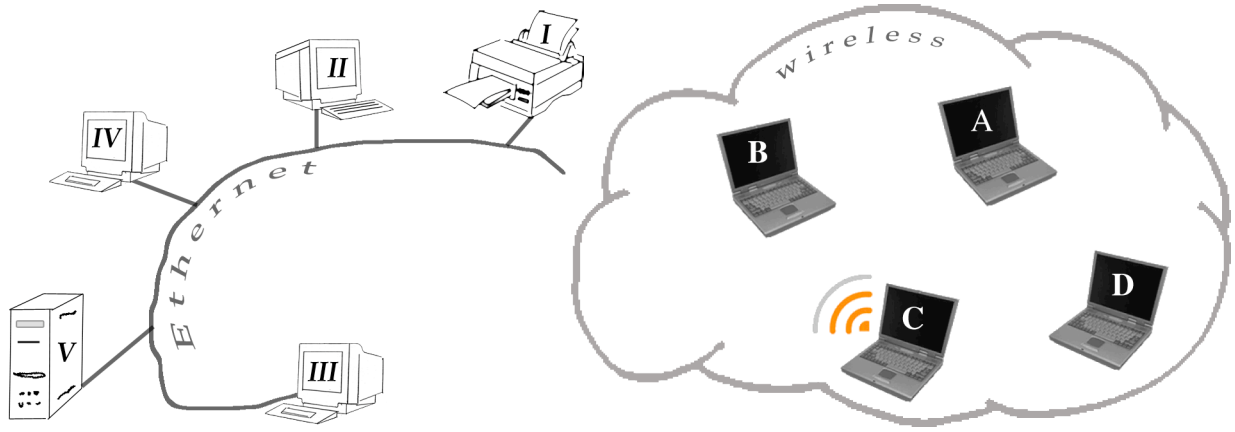

Figure 7.1: Two networks using distinct communication technologies

you should understand many mysterious terms you have probably encountered when trying to configure your computer's network settings including "DHCP" and "subnet mask". In the process, you will learn quite a bit about how the Internet actually works.

7.1 Routers

To keep things simple, let us start by considering the problem of providing communications between computers on just two distinct networks, a wired Ethernet and a wireless network. Figure 7.1 shows a diagram of what we have in mind. Everything on the left side of the figure is identical to a diagram that was presented in Figure 6.8 to illustrate the structure of Ethernets. For our purposes, we might imagine that this network connects computers in the office of some small business. The computers labeled II, III, and IV might be office desktop machines. Computer V might be the company's server, and I is a shared printer.

Now imagine that several of the company's employees use laptop computers rather than desktop machines and that these laptops are equipped with wireless network adapters rather than with Ethernet adapters. The laptops will then form a separate network as shown on the right side of Figure 7.1. They will be able to communicate with one another, but unable to access the shared printer and server connected to the Ethernet on the left side. Obviously, the company using these networks would be eager to find some way to provide inter-network communications.

In our discussion of wireless networks, we already introduced one important component of an internet, the *router*. A router is a computer that is part of two or more distinct networks and is configured to relay messages between the networks to which it is connected. In our discussion of wireless networks, we referred to the device that acted as a router as a hub, since that is the name commonly used in the context of a wireless network. Router is a more generic term applicable even when none of the networks being interconnected is wireless.

Figure 7.2 shows a router installed between the Ethernet and wireless networks from Figure 7.1. The idea behind routers is simple. Suppose computer A, one of the machines connected through the wireless network in the figure, wants to send a document to the printer labeled I. Since the printer is not on the wireless network, A cannot send messages containing the document directly

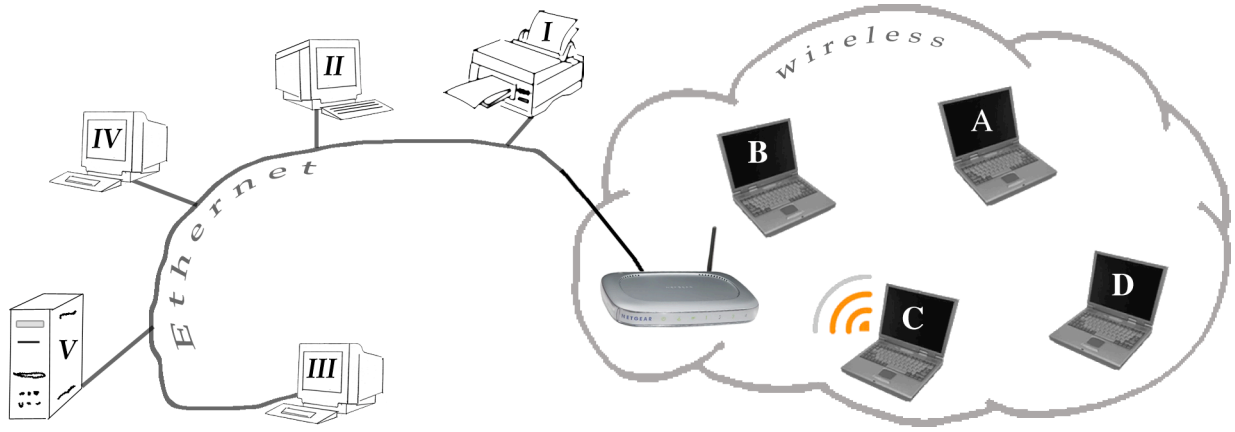

Figure 7.2: Two networks interconnected by a hub

to the printer. Instead, it will send these messages to the router. The messages must be sent to the router in a special format indicating that they are not really meant for the router. Information provided in this special message format will also enable the router to determine the desired final address and send the message to the printer.

In Figure 7.2, we have used a picture of a popular wireless hub to represent the router. The physical packaging of the hub and the "rabbit ear" appearance of its incoming Ethernet cable and wireless antenna clearly distinguish the router from other devices connected to the two networks in the diagram. From the point of view of the networks, however, all of these devices are very similar. They are all computers that can send and receive messages through the network. Based on the software installed on these devices, they will send different kinds of messages and react to the messages they receive in very different ways. The key point, however, is that a router is basically a computer.

In fact, the software provided with most current desktop and laptop computers can easily be configured to make a computer act as a router (as long as the machine is connected to multiple networks). Figure 7.3 shows the dialog boxes used to configure a computer as a router under Windows XP and MacOS X.[1] The window on the left is a Windows XP control panel. Clicking in the checkbox labeled "Allow other network users to connect through this computer's Internet connection" tells the computer's software that it should act as a router between the two networks with which it can communicate. The window on the right is an Apple MacOS X system preference panel. Clicking "Start" in this panel tells the computer to start acting as a router between the "Built-in Ethernet" and the computer's "Airport" wireless network interface. As a result, it would be perfectly reasonable to replace the diagram in Figure 7.2 with the one shown in Figure 7.4 in which we show a general-purpose laptop computer configured to act as the router.

When a router forwards a message from one network to another it may have to do quite a bit of work. Each network comes with its own protocols that dictate many details of the process of communications. These protocols typically include precise specifications for the format of packets. They dictate the order of the fields in a packet. In one protocol, the return address might come

[1]Many network administrators frown on users who configure their computers to act as routers. If you have the urge to experiment with the mechanisms illustrated in Figure 7.3, and your computer is connected to a corporate or institutional network, you might want to check with the computing staff beforehand.

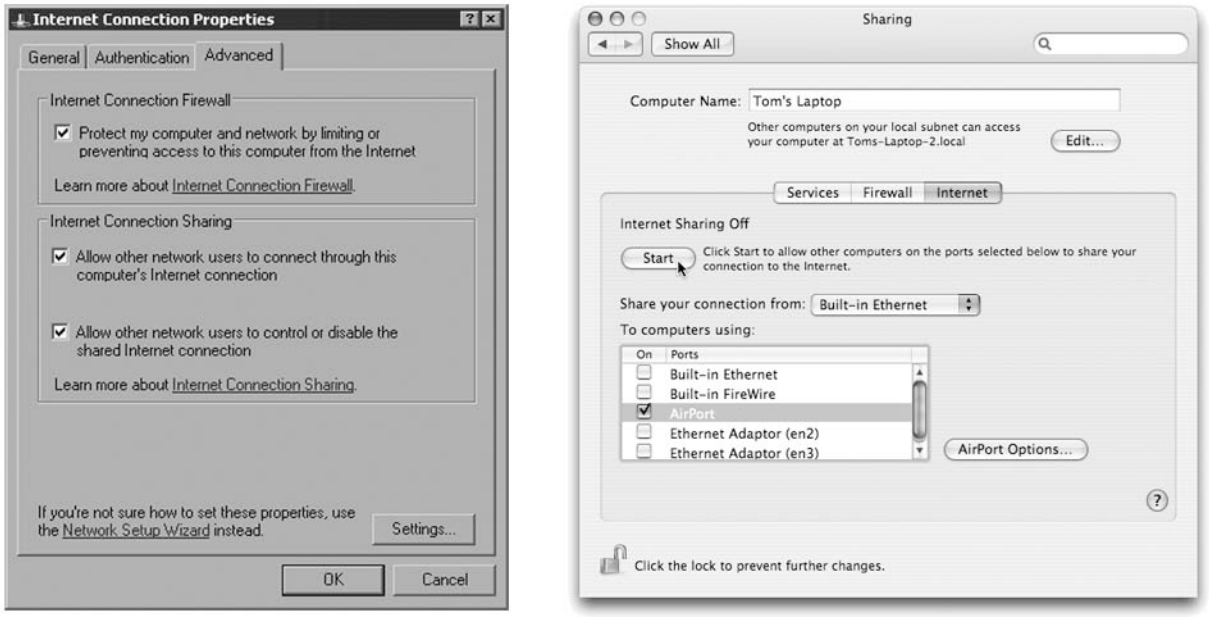

Figure 7.3: How to turn your computer into a router

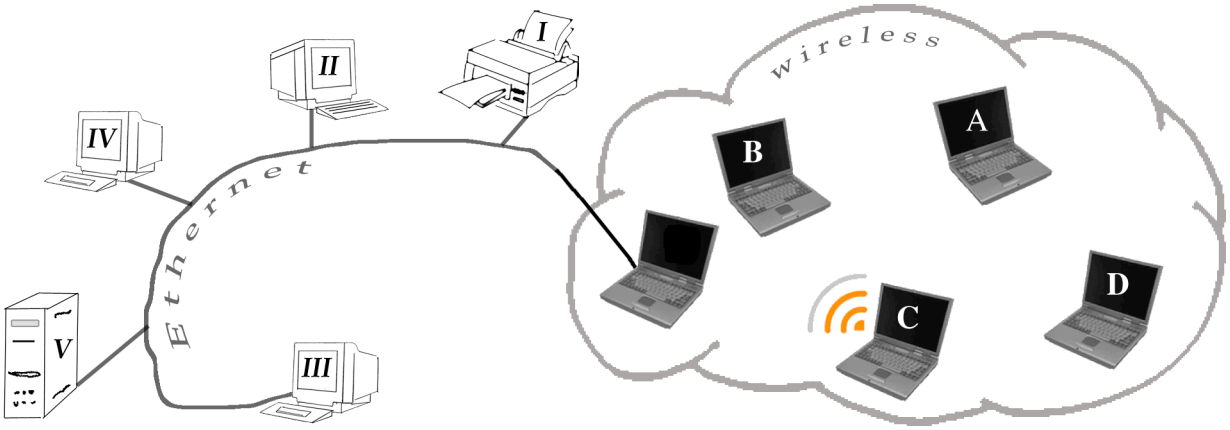

Figure 7.4: A laptop functioning as a router

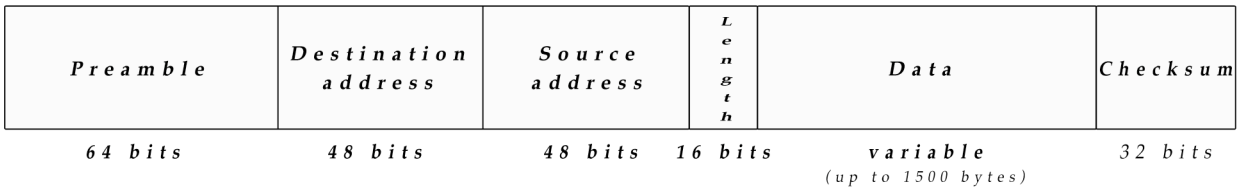

Figure 7.5: Ethernet packet format

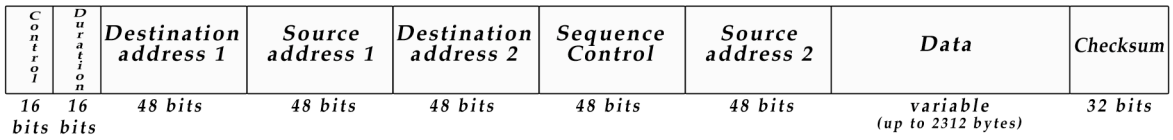

Figure 7.6: Frame format used by 802.11 wireless networks

before the destination address while in another the destination address might come first. Fields that are required in one protocol may be absent in another or at least of a different size.

If Figure 6.10, we showed the format used when packets are transmitted on an Ethernet. For convenience, that diagram is repeated in Figure 7.5. To enable you to appreciate how packet formats can differ, in Figure 7.6 we show the format of packets sent on a wireless network based on the widely used IEEE 802.11 protocol specification. Compare these two packet formats. You should notice many differences.

The first difference you might notice is that there is no preamble field at the beginning of the wireless frame. This is misleading. The wireless frame does include a preamble. We omitted this field from our diagram for two reasons. First, the wireless frame has many more fields than the Ethernet frame. We really didn't have enough space to show them all. Second, the format of the wireless preamble varies depending on the transmission technique used. If we wanted to describe it accurately, we would have to show several variants of the packet format. In all cases, however, the length of the wireless preamble is greater than the 64 bit preamble used by Ethernet.

You should also notice that the wireless packet has two fields labeled "Control" and "Duration" that are not present in the Ethernet frame. These fields are essential to the process wireless networks use to approximate carrier sense. The first field contains information used to distinguish the special RTS and CTS packets from data packets. The second is used to hold transmission duration information essential to the process of "virtual carrier sense". It is not critical that you recall all the details of these mechanisms at this point. What is critical is that you understand that these differences between the Ethernet and wireless packet formats are not accidental or capricious. They are the result of fundamental differences between the transmission techniques used by these network technologies.

These differences imply that the task a router must perform is non-trivial. It cannot simply take the packets it receives from one network and resend them to their ultimate destinations on the other network. Instead, some form of translation is required that accounts for the peculiarities of the distinct protocols used by the networks to which the router is attached.

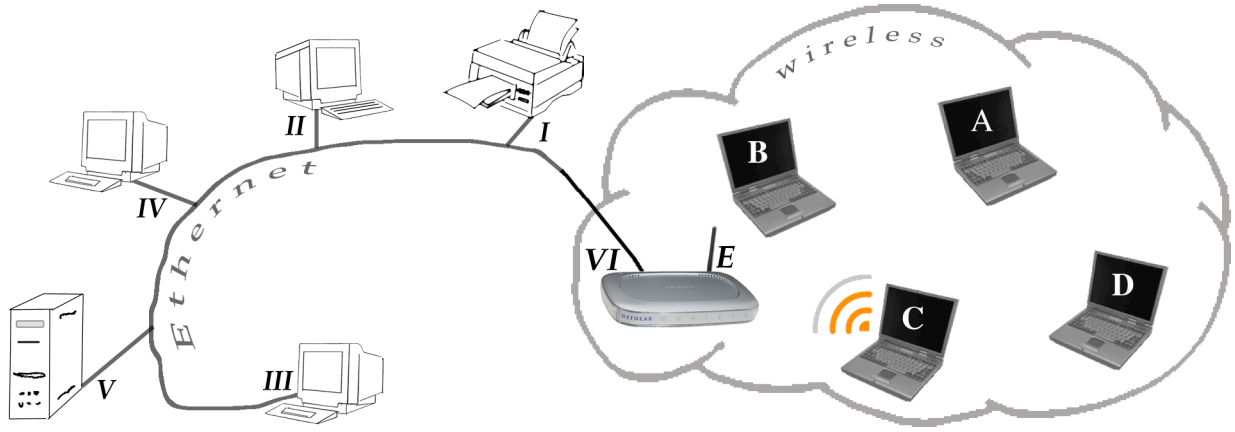

Figure 7.7: A router requires distinct addresses for each network interface

You have also probably noticed that the wireless packet format has two more address fields than the Ethernet packet. We will not attempt to explain why wireless packets require four addresses rather than two. We do, however, want to talk a bit about addresses.

In Figures 7.4 and 7.2 the laptops are labeled with letters while the computers on the Ethernet are labeled with Roman numerals. We deliberately used distinct sets of labels for the computers shown on the two networks to suggest one important fact about networks. Different networks may use different formats for machine addresses and, even if they use the same format, addresses used on one network will generally be meaningless on other networks. For example, while both Ethernet and the 802.11 wireless standard use 48 bit addresses, there are standards for switched networks such as Frame Relay and HDLC that use 8 and 16 bit addresses.

The fact that the addresses used in one network are distinct from those used by other networks implies something interesting about routers. Suppose that the two networks in Figures 7.4 used distinctly different address formats. The Ethernet might use 48 bit addresses while the wireless network used 16 bit addresses. Which type of address should be associated with the router? The answer is both! In order for one of the laptops to send a message to the router through the wireless network, it has to use an address that is compatible with the wireless network protocol. On the other hand, in order for one of the desktop machines to send message to the router through the Ethernet, the router must have a 48 bit Ethernet address.

This reveals an important property of a network address. A network address does not actually identify a machine. It identifies the connection between a machine and a network. Each cable or antennae connecting a machine to a network logically has a distinct address. In particular, because routers are connected to multiple networks, a router will always have multiple addresses. Given this understanding of addresses, Figure 7.7 presents one more illustration of our two-network internet in which the router is labeled with both a letter and a Roman numeral to reflect the fact that each of its network connections will require a distinct address.

7.2 An internet Protocol

As we emphasized in the preceding section, routers are just computers. To function appropriately, all computers need software. Therefore, we need more than the physical hardware of a router to have a functioning internet. We also need to implement and install appropriate software in both the routers and the computer connected to our internet. The issues that arise when constructing this software lead to a key component of the internet concept: the need for an internet protocol.

Imagine that you are the user of laptop B in Figure 7.7. Each time you click on a link in a web page, the software that makes up your web browser application has to create a sequence of bits that encodes a request for the desired web page and transmit those bits through the network. When this message is sent, it has to be encoded in the format appropriate for the wireless network. This would appear to require that your web browser include software designed specifically to format data in the manner required by the wireless network protocols. The same, of course, would be true of your IM chat client, your email program, and any other network-related software you might use.

Even if your laptop usually communicates using the wireless network, chances are that somedays you might plug the machine into the Ethernet either for extra speed, privacy, or some other reason. Suppose you continue using your web browser after plugging into the Ethernet. Now when you click on a link, the same program has to format the requests it sends in the manner required by the Ethernet. Apparently, your web browser's code would need segments specifically designed for both the wireless and Ethernet protocols. In fact, unless you are willing to update your browser every time you use a new network, your browser should include segments of code that can handle all of the formats of all of the network technologies you might ever use. By similar logic, this would also apply to your IM client, your email program, etc. This could require writing a lot of software, and writing reliable software is difficult and expensive.

The situation looks even worse if we think about the software required for routers. As we have indicated, a router must have the ability to translate packets from the format required by one of the networks to which it is connected to the format(s) of the other(s). Figure 7.8 shows a logical view of what goes on "underneath the hood" in a typical router. The figure attempts to differentiate those aspects of the router's functionality that are embedded in its physical hardware from those functions described by software. The hardware present in the router provides the ability to send and receive packets on both networks. The ability to translate between formats, however, is provided by the software installed. Note that the translation software must be capable of two-way translation. That is, it must be able both to translate an Ethernet packet into a wireless packet and to translate a wireless packet into an Ethernet packet.

The exact software required for each router will depend on the types of the networks it is supposed to interconnect. At this point, we have only talked about two specific physical network protocols, Ethernet (also known as IEEE 802.3) and the IEEE 802.11 wireless standard. There are many more. There are protocols based on broadcasting other than Ethernet and wireless. There are a variety of standard protocols for switched networks. Then there are standards for networks based on satellite transmission, for data services to cell phones, etc.

In general, a router may be connected to 2, 3, or more networks using different protocols. To simplify our discussion for a moment, let us pretend that all routers are connected to just two networks like the router shown in Figure 7.7. Even in this simple case, for each type of network, we would potentially need router software to translate between its format and every other format. We would need an Ethernet to wireless translator, an Ethernet to Frame Relay translator, a Frame Relay to wireless translator, a Token Ring to DBDQ translator, and so on. Even if we limit our

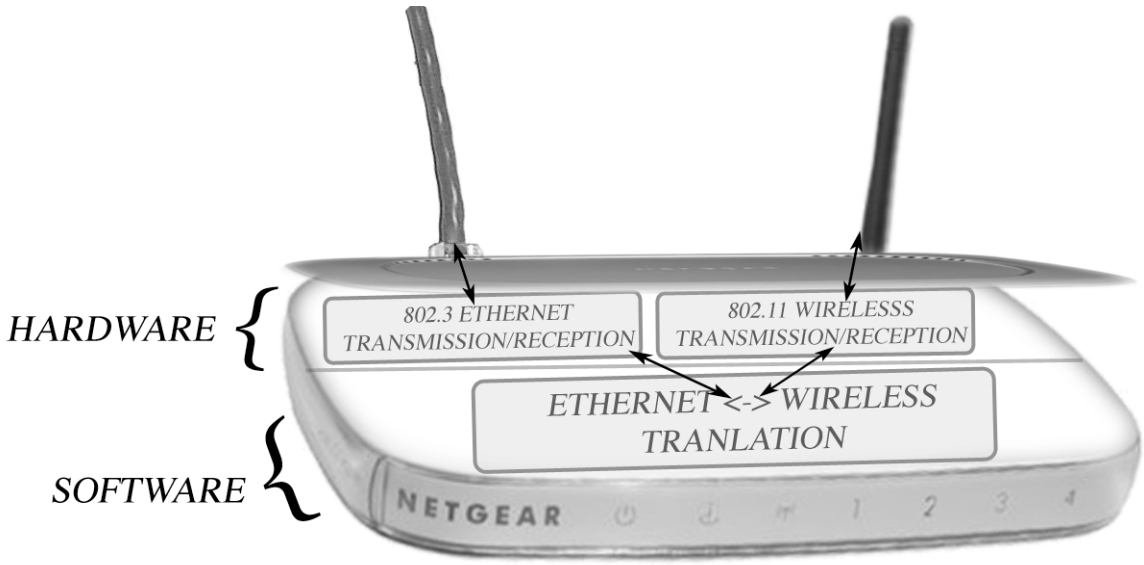

Figure 7.8: Software and hardware components in a router

routers to connecting pairs of networks, if there are N different protocols, there are $\frac{N(N-1)}{2}$ pairs for which we need translators. If N is 10, that is 45 pairs. If N is 20, there are 190 pairs.

It appears that the larger the number of distinct types of networks we want to interconnect, the more software is required both in clients and in routers. Surprisingly, the technique that makes the software requirements manageable is to design one more protocol. This extra protocol is called an internet protocol.

Just like the other protocols, the internet protocol will have its own packet format, its own way of addressing computers, and its own rules for how to send and receive packets. It will, however, serve a very different role from the other protocols. It will serve as an intermediate language used to simplify the construction of the software required to translate between other protocols. Within the Internet, the protocol designed for this purpose is cleverly named the "Internet Protocol" and called IP for short. For now, everything we are saying applies to the general notion of *an* internet protocol rather than the concrete details of *the* Internet Protocol. Therefore, we will use the short name iP when we talk about the properties of a generic internet protocol.

Above, we suggested that we would need router software that described the steps required to translate between any pair of protocols used on the networks we seek to interconnect. Suppose we start the process of implementing this software by writing instructions describing how to translate between iP and all of the other protocols. We would write an Ethernet to iP translator, a wireless to iP translator, a Token Ring to iP translator, a Frame Relay to iP translator, and so on.

Once this collection of translators is complete, there is an easy way to construct a translator for any other pair of protocols. Suppose, for example, that we wanted to construct a translator for the router in Figure 7.8, an Ethernet to wireless translator. Recall that translators are two-way devices. The software written to provide an Ethernet to iP translator must both be able to convert an Ethernet packet into iP format and to convert iP format packets into Ethernet format.

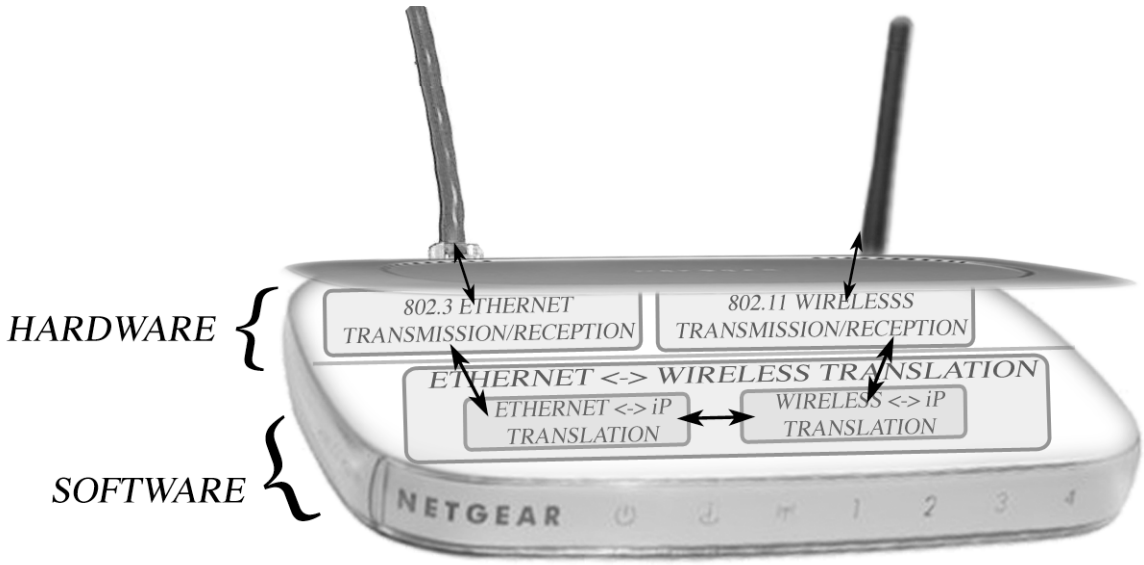

Figure 7.9: The internet protocol format serves as an intermediate language

As a result, we can construct an Ethernet to wireless translator by simply pairing together two our our iP translators. In particular, we would use an Ethernet to iP translator and a wireless to iP translator. The resulting software will translate from Ethernet to wireless by first translating from Ethernet to iP and then translating the result from iP to wireless. Similarly, it can translate from wireless to Ethernet by first translating from wireless to iP and then translating from iP to Ethernet. Suddenly, we can handle $\frac{N(N-1)}{2}$ possible pairs of network types after writing software for only N translators.

Figure 7.9 shows a view of the logical components of a router, refined to reflect the fact that the translation software can actually be composed of two software modules designed to work with the internet protocol. The data that travels along the path represented by the two-headed horizontal arrow in the center of the figure will be encoded in the iP packet format. This is how iP serves as an intermediate language. Data traveling along the vertically-oriented two-headed arrows on the left side of the figure will be encoded using the Ethernet packet format, and data traveling along the two-headed arrows on the right side will be encoded in the wireless packet format.

The same collection of iP-to-everything-else translators can also be used to simplify the task of constructing client programs. Suppose that when writing a network client program like a web browser, we write all the code to format the messages that our program sends using the iP format and we assume that all the messages it receives will arrive in the iP format. Then, if the program is run on a computer that is actually connected to a wireless network, all we need to do is pair up the browser's code with the code for our iP to wireless translator. Whenever the browser tries to send an iP packet, we will first run the packet through the iP to wireless translator to obtain a version of the message encoded in the format required by the wireless network. Similarly, whenever a packet for the browser arrives through the wireless network we will run it through the iP to wireless translator so that we can deliver it to the browser in the iP format it expects.

The introduction of an internet protocol greatly simplifies the task of constructing software to support inter-network communications. We do not need translators for every pair of possible network types. Instead, we just need one translator for each network type that can translate between that network's packet format and the internet protocol format.

7.3 The Role of the Operating System

Figure 7.10 shows the familiar features of the two widely used operating systems, Windows and MacOS X. Everyone knows that you must have an operating system installed on your computer, but it is not clear to many users what an operating system actually is and what exactly it does.

We know that the operating system provides the interface that we use when we put our fingers on our mouse or keyboard. We also know that the applications we install on a computer have to be compatible with the operating system used on that machine. Figure 7.11 displays a familiar choice. The figure shows part of the AOL web page from which one can download AOL's instant messaging client. There are different clients for various operating system: one for Windows, one for MacOS, and one for Linux. Only the correct version will run on your system.

We take this for granted, but it is actually a bit strange. The computers that run Windows, MacOS X, and Linux are all now based on the same processor chips from Intel (or equivalent chips from Advanced Micro Devices). MacOS X is running on my computer as I type this text, but I can also run Windows on the same machine. If my computer contains all the hardware needed to run the Windows version of AIM under Windows, why can't it run the Windows version of AIM while running under MacOS X? To answer this question, we have to understand how a program interacts with the operating systems on a computer.

The operating system in your computer is itself a program. It is clearly special in several ways. Unlike other programs, you do not have to double-click on some icon to make the operating system start running. It starts as soon as you turn the machine's power on and continues to run (hopefully) as long as you continue to use your machine. Like all of the other programs on your machine, however, the operating system is just a long sequence of instructions that determine what the computer's hardware will do in response to actions you perform with the mouse and keyboard and in response to other input the machine may receive (including messages received through the network).

Many of the applications we use on our computers manage collections of various kinds. Your email client manages your collection of email messages. It displays summaries of the messages in the collection. When we click one of these summaries with the mouse, it responds by displaying the associated message. A word processor manages a collection of letters or symbols. Again, the mouse can be used to indicate which part of this collection we wish to access or modify. You might also use programs that manage collections of photos, collections of audio files, etc.

It is possible and useful to understand your operating system as a collection manager. It manages the collection of programs installed and running on your machine. By double-clicking on a program's icon, you tell the operating system that you want that program to start running. When this new program starts, it becomes the *active program* on your computer. In general, the program associated with the top-most window on your screen is usually considered the *active program* on your computer. It is the program that receives text typed on your keyboard.

You control which program is active by communicating with the operating system. You do this by clicking the mouse in an particular region on your computer's display. When you click within

Figure 7.10: Your operating system provides a user interface

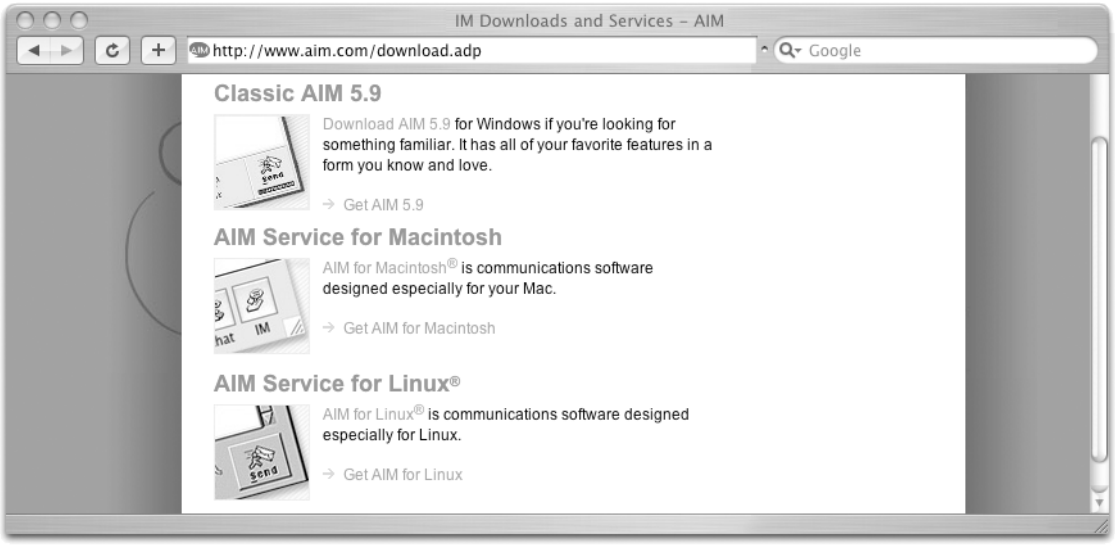

Figure 7.11: Application programs are operating system dependent

the boundaries of the top-most window on your desktop, the active program responds. You are communicating with the active program rather than with the operating system. If you click on any other window, however, you are communicating with the operating system. By clicking on the window of a program that is not currently the active program, you tell the operating system that you want that program to become the active program.[2] By clicking on a "finder" window, you tell the operating system that you want the operating system to become the active program, probably so that you can tell it to launch another program by double-clicking on some document icon.

7.3.1 Sharing Nicely

As a "program manager" the operating system not only allows you to control how you interact with the program's running on your computer, it is also integral to supervising how those programs interact with one another and with the computer's hardware. In particular, it makes sure that all of the programs running on your computer share nicely. This includes sharing the computer's hardware resources. Among others, the operating system manages the sharing of your computer's processor, its memory, its keyboard, its display, and even its network connection. In addition, the operating system provides a convenient way to share software components among multiple programs.

As an example of sharing hardware, consider the keyboard. Most programs that you run on your computer respond to some form of keyboard input. When you are running 10 programs at the same time, however, you don't need 10 keyboards. You only need one. When you type text, the system somehow has to arrange for the correct application to receive and respond to the text you type. The hardware components in your keyboard and computer do not know enough to do this

[2]On some systems, clicking is not required to make a program active. Merely moving the mouse cursor into a program's window makes that program the recipient of keyboard input. To keep our discussion simple, we will ignore this interface option. The key point, however, is that with both approaches you determine which program is active by using the mouse to communicate with the operating system.

on their own. The hardware components only do what the instructions in the software tell them to do. Somewhere in your computer's software, there have to be instructions that determine how the keyboard input gets to the correct application.

As we have seen, the operating system determines which program is active based on user actions with the mouse. As a result, the instructions that determine which running program should receive keyboard input must be part of the code of the operating system. Therefore, whenever keyboard input is received, the hardware first follows instructions found in the operating system to determine how the text should be handled. These instructions then direct the hardware to continue by following the instructions for responding to keyboard input within the active program.

This is typical of many of the relationships between the operating system and other programs on a computer. Almost all direct interactions between hardware components and software running on your computer involve software that is part of the operating system. This applies not only when the machine receives input from the keyboard, the mouse, or some other device. It also occurs when a program tries to display some output.

Suppose a program decides to display some text on the screen. We don't expect a program to be able to display such text anywhere on the screen. We expect all text displayed by a given program to appear in that program's window. We would be disturbed if we clicked on a link in our web browser and the contents of the new web page appeared in our word processor window.

If programs were allowed to send commands directly to the computer's display, then an incorrect program could easily send commands that changed the information displayed within another program's window on the screen. Most systems prevent this by only allowing the operating system to actually send commands to the display. If an application program wants to display anything, it must ask the operating system to do the actual work of placing the new information on the screen. The application program must contain instructions that construct a description of the information to be displayed and instructions to pass this description to the operating system. The instructions that actually place the new image on the display will be part of the operating system software.

This approach both prevents accidents and makes it easier to write application programs. To display information, all an application program has to do is describe what it wants to display. It does not have to keep track of where on the screen its window has been placed. The operating system will do that and place the new information in the right portion of the screen. The application program also does not need code to determine whether parts of its window are obscured by other windows on the screen. These instructions will be part of the operating system software.

This illustrates how an operating system facilities both the sharing of hardware and software. The operating system is obviously managing the sharing of the display hardware. In addition, all of the application programs running on the machine are sharing a single copy of the instructions that tell the computer how to manage windows on the screen. These instructions are part of the operating system. As a result, one copy of the instructions can be shared by all applications.

It also makes it possible to explain why a program written to run under MacOS X will not run under Windows and vice versa. When a program asks the operating system to display some new information on the machine's display, the program must provide the operating system with a description of the information to be displayed. This description has to be encoded using a scheme that both the program and the operating system understand. The encoding scheme used will be part of a protocol or standard that specifies the interface between application programs and the operating system. The details of such encoding schemes and many other aspects of the application program's interface with the operating system vary from one system to another just as the details

Figure 7.12: The operating system controls program interactions with keyboard and display

of communication protocols differ from one network to another. An application program will only work on a particular operating system if it adheres to the standard describing that operating system's programming interface.

Figure 7.12 illustrates the logical relationships between application software, operating system software, and the two hardware components we have used as examples, the keyboard and display. The figure shows a computer running the same applications shown in the screenshots in Figure 7.10, a web browser, a calculator, and the Skype voice-over-IP client. The boxes drawn within the processor box indicate how the software controlling the machine can be divided into subcomponents. Each of the applications is a distinct component, and they are all separate from the code for the operating system. Within the operating system, there are subcomponents composed of instructions that deal with the display hardware, deal with the keyboard, and manage all of the application programs and their windows. The arrows in the diagram show some of the paths along which information flows among these components. Keyboard input is processed by components of the operating system that determine which application should receive the input and then forward the input to that application. Application requests to display new information on the screen are first sent to the operating system. The operating system determines where the information belongs on the screen and then sends it to the display hardware.

7.3.2 Sharing Network Resources

With this understanding of the basic relationships between application programs and an operating system, we can explore the role the operating system plays in the implementation of the internet concept.

Just as all the programs running on a single computer must share a single keyboard, many programs often share a single network connection. The computer shown in Figure 7.12 is running two network applications, a web browser and Skype. When a message arrives through the computer's

single hardware connection to the network, some component of the software on the computer has to determine whether the incoming message is for the browser or for Skype and deliver it to the right program. Just as a component of the operating system directs keyboard input to the right application program, the task of directing network messages to the appropriate application is handled by networking software within the operating system.

Of course, the format of incoming network messages will depend upon the type of the physical network to which the computer is connected. If the computer is connected to an Ethernet, incoming messages will arrive in Ethernet format. If a wireless network is being used, messages will arrive in wireless format. In Section 7.2, we suggested that all incoming messages should be translated into the common internet protocol format before being processed by applications. Now we can see where the code that performs this translation should reside. All incoming network messages must pass through the operating system. If the operating system includes the code for translating between the format used by the network to which a machine is physically connected and the internet protocol format, then all incoming messages will be encoded in the internet protocol format by the time they reach an application program.

In our discussion of how programs share a single display, we saw that a program that wants to place information on a computer's display actually has to ask the operating system to place the desired information on the display. On most systems, requests to transmit messages through the network are handled in a similar way. Only the operating system is allowed to directly interact with the hardware that actually transmits messages through the network. When an application program wants to send a message, it has to ask the operating system to do the actual transmission.

Now recall that in Section 7.2, we assumed that the software modules that translated between the internet protocol format and other network packet formats would be capable of two-way translation. If the operating system includes such translation software, then when an application gives the operating system a message to send, the application can describe the message using the internet protocol format and leave it up to the operating system to translate the message into the appropriate format for the actual network being used. This makes it possible for programs to both share a single network connection and to share a great deal of code. Rather than placing copies of the instructions that deal with the packet formats of each network in each network application, all network applications can share the translation software found within the operating system. Figure 7.13 depicts the logical structure of these software components.

If the operating system handles translation to and from the iP format in this way, application programs can be written as if all machines are connected to a network based on the internet protocol even though each machine is actually connected to some particular physical network with its own peculiar packet format. This is remarkable. In some sense, the internet protocol isn't real at all. All messages actually traveling through network wires must be encoded in the format of some other protocol. The operating system, however, gives programmers the illusion that all network messages are encoded in the internet protocol format. To these programs, the internet appears to be a real network incorporating all of the physical networks it interconnects.

As we continue our discussion of internetworking, we will frequently need to distinguish the real networks through which packets travel from the illusion that all of the networks interconnected by routers form a single network. We will refer to the combined network as an internet or a *virtual network*. We will refer to the independent network that make up this internet as *hardware networks* or *physical networks*.

Leaving the task of translating between packet formats to the operating system has one final

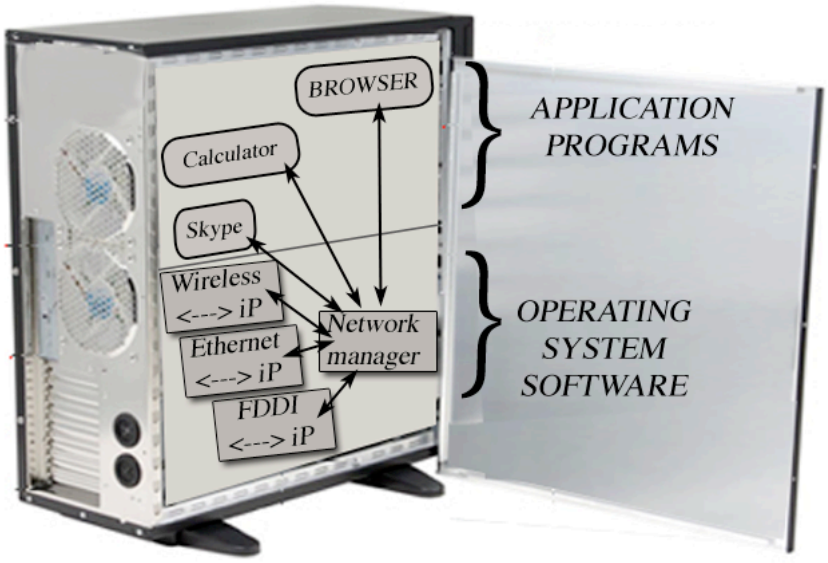

Figure 7.13: The structure of an operating system's network software components

benefit. Application programs can continue to function smoothly even when a computer is moved from one network to another. It is not uncommon for a laptop user to work for several hours using a wireless network and then to plug the machine into an Ethernet at some other time. If iP translation software for both the Ethernet and wireless protocols is included in the machine's operating system, then changing networks can be handled with no impact on application programs. The operating system can sense which network hardware components are active. When it senses an active wireless network, it can translate outgoing iP packets to wireless format and transmit them through the air. When it senses a cable is plugged into the machine's Ethernet jack, it can translate iP packets to Ethernet format and send them through its Ethernet cable.

7.4 Lost in Translation

We have now discussed the need for software to translate between protocol formats and the relationship between these software modules, application software, and operating system software. We have not, however, discussed how these software modules will accomplish such translations. In this section, we will present a simple technique that makes the translation process almost trivial.

Up to this point, we have tried hard to talk about the internet concept in a general way. Rather than talking about the details of THE Internet and THE Internet Protocol we have talked about internets and an internet protocol. To explain how to translate between hardware packet formats and an internet protocol format we need a specific example of an internet protocol format. A natural choice is the format used by the most widely used internet protocol, IP, THE Internet protocol. At this point, therefore, we will begin to focus on how the Internet protocol is implemented.

The packet format used by IP is shown in Figure 7.14. The diagram in this figure uses very different stylistic conventions than those used to show packet layouts in Figures 7.5 and 7.6. In the earlier diagrams, the fields of the packet were presented on a single line drawn from left to right in

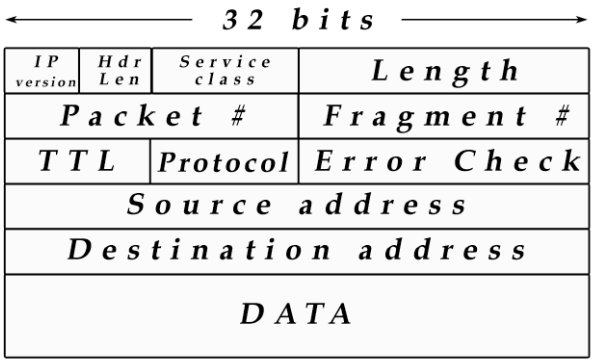

Figure 7.14: The format of an IP packet

the order in which their bits would be transmitted on the network. In Figure 7.14 the fields are drawn on several lines ordered from left to right and from top to bottom in the order that they would be transmitted. That is, the "IP version" field would be sent first, followed by the "Hdr Len" (Header Length) field, the "Service Class" field, and the "Length" field. These would then be followed by the "Packet #" field and so on. Also, in the earlier packet diagrams, the size of each field was specified directly under the field. In Figure 7.14, field lengths must be determined by scale. The entire width of the stack of fields represents 32 bits. Therefore, a field that appears by itself on a level, like "Source Address", corresponds to 32 bits of data. On the other hand, a field like "Service Class", which occupies just one quarter of a level, corresponds to 8 bits of data. The "DATA" field is the exception to this scaling rule. It is not limited to 32 bits. In fact, IP packets can hold up to 65,536 bytes of data. These are the standard conventions used for describing packet formats in the Internet standard documents known as RFCs (RFC is short for "Request for Comment").

Beyond the different formatting conventions used in these diagrams, there are substantive differences between the formats used by IP, Ethernet, and wireless networks. These differences are easy to see as you examine Figures 7.5, 7.6, and 7.14. IP uses 32 bits for addresses while the other protocols use 48 bits. IP packets have several fields ("Service Class", "TTL", and "Fragment #" are examples) with no clear equivalent in the other formats. Nevertheless, we need some way to translate between the IP format and these and other hardware network packet formats.

In one sense, this problem is even harder than it may already seem. The translation process we use must be perfectly reversible. When a network application program sends a message, it will format it as an IP packet. This packet will then be translated into the format of the network to which the machine is attached. Assuming it is sent to another machine on the same network, it will next be translated back into IP by the operating system on the destination machine and delivered to an application. The packet received by the destination application must be identical to the packet sent by the source.

Such perfect reversibility is not usually the case when translating between human languages. For example, Google provides a nice translation service through its web site. As shown in Figure 7.15, Google translates the title of this section "Lost in Translation" into Korean as "손실의 번역". If you then enter the Korean phrase "손실의 번역" and ask Google to translate this back into English, the result is "Lost in the translation." This is very similar, but not identical to the original phrase.

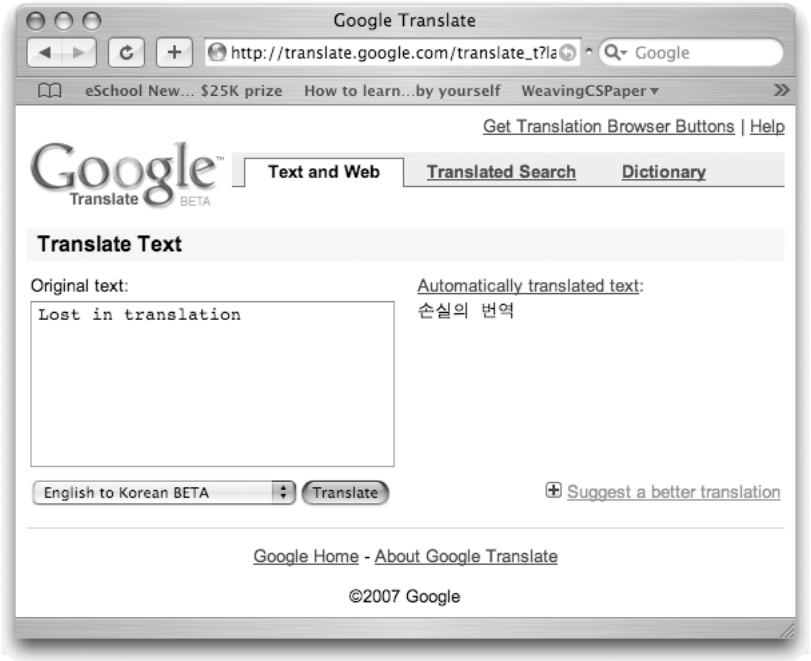

Figure 7.15: Google's web translation service

Unsurprisingly, Google does much worse on many other examples. For example, if you translate "Lost in Space" into Korean and back again, the result is "Loss of space", a phrase with a very different meaning. In general, the English-to-Korean translation process is not perfectly reversible. This would not be acceptable behavior for an IP translation scheme.

The surprising thing is that there is a simple way to accomplish such perfectly reversible translation. Look back at all of the packet format diagrams we have presented. One thing that they all have in common is a field to hold the actual data of the message being sent. The rest of the fields are mainly addresses or other data describing how to handle the information in the data field. In this sense, a packet is a bit like an envelope. With an envelope, we put addresses on the outside and a message, the data, on the inside.

Envelopes and packets have something else in common. An envelope is made of paper, and we usually put messages written on paper inside. In fact, you can take one envelope and, possibly with a bit of folding, fit it inside another envelope even if the first envelope already contains a letter. Similarly, packets formats describe sequences of binary digits, and the messages we put in the data fields of a packet are also sequences of binary digits. This means that (with the possibility of some sort of "folding") we can take one packet and fit it into the data field of another packet using a different format, even if the first packet already contains data of its own. This process is called *encapsulation*.

Figures 7.16 and 7.17 show what we have in mind. The first figure shows the contents of an IP packet placed in the data field of an Ethernet packet. The second is very similar. It shows an IP packet encapsulated within the data field of a wireless packet. This provides the basis for a way to translate IP packets into hardware protocol formats. You simply place the bits that represent the IP packet in the data field of a packet of the desired hardware format. It is obviously easy to

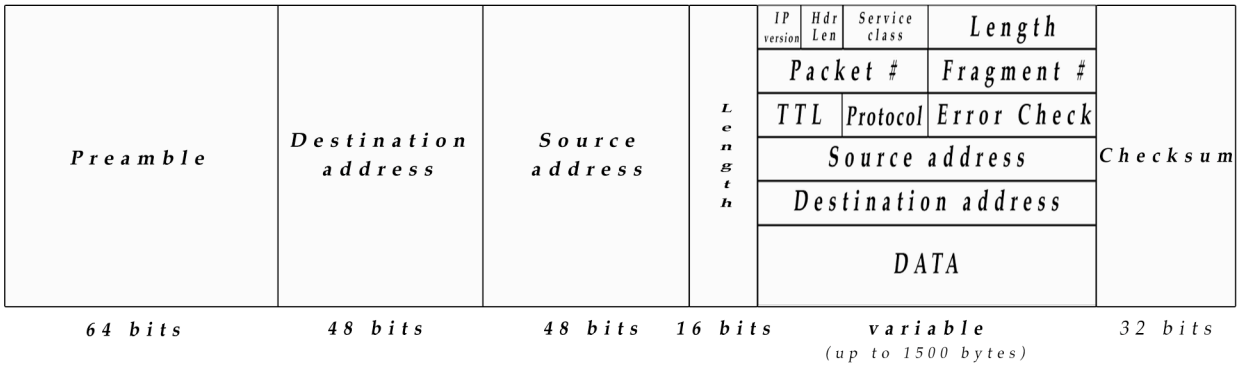

Figure 7.16: An IP packet encapsulated as the data of an Ethernet packet

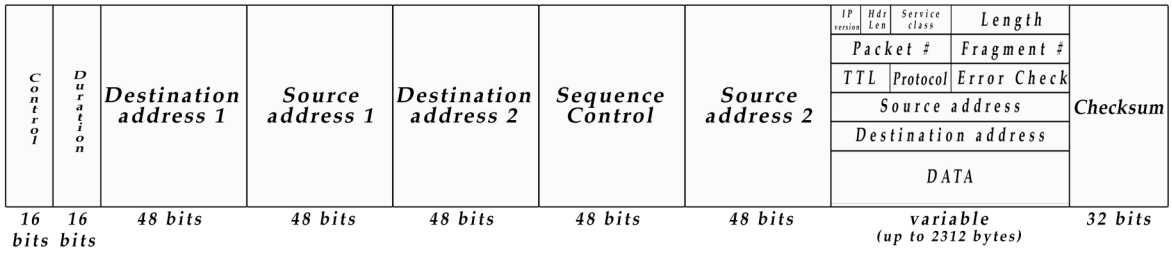

Figure 7.17: An IP packet encapsulated as the data of an 802.11 wireless packet

translate such hardware packets back into IP format. You simply pull out the data field. Best of all, the translation process is perfectly reversible as required.

There are two aspects of this approach to translating between packet formats that are not quite as trivial as we have tried to make them appear. First, looking at the diagrams in Figures 7.16 and 7.17, you might notice that the Ethernet and wireless protocols put different limits on the amount of information that can be placed in the data field of a packet. Other hardware protocols have different and in some cases smaller limits. What happens if the IP packet we want to send won't fit in the data field of a given hardware protocol packet. How do we "fold" an IP packet to make it fit in a small data field?

In theory, this problem is easy to solve. While you cannot fold an IP packet, you can cut the data it contains into smaller pieces that will fit in hardware packets, send these pieces in separate hardware packets and sew them back together for delivery as a single packet within the operating system of the receiving machine. This process is called *fragmentation*. The technical details of fragmentation are not worth examining at length here. We will simply note that the "Packet #" and "Fragment #" fields shown in the IP packet format are used to enable the receiving computer to identify and reassemble the fragments of a single message.

There is one other more fundamental issue. Encapsulation gives us a great way to fill the data field of a hardware packet and it ensures that we can retrieve the original IP packet exactly from

the data sent through the network. It does not, however, explain how to fill in the components of the hardware packet other than the data field. In particular, we need a way to translate the IP source and destination addresses in an IP packet into Ethernet or wireless addresses. This task will be the underlying subject of the remaining sections of this chapter.

7.5 Change of Address

Before we think about translating IP addresses into Ethernet addresses or any other format we should learn a bit more about the formats of these addresses and about network addresses in general. A good place to start is by appreciating some properties of various forms of "addresses" you might find in your address book including street addresses, IM buddy names, and even phone numbers.

7.5.1 Forms of Address

The first advantage of taking a broad view of addresses is that it becomes clear that addresses do not just identify places. An address is just a name for something. A street address names a place, an IM buddy name identifies a person, a network address identifies a connection between a computer and a network.

One important property of addresses is that they must be unique. If I tell you that my home address is 40, that does not help you very much. Even telling you that my address is 40 Talcott Road is not enough since there are streets named Talcott Road in Massachusetts, Connecticut, New Jersey and British Columbia. I have to provide more information to make my address unique. I have to tell you that my complete address is 40 Talcott Road; Williamstown, Massachusetts.

There are at least two approaches to ensuring that addresses are unique. One is to establish some central repository of addresses and insist that each new address be checked against those in the repository to make sure that it is not already being used. IM screen names work this way. When you go to the AOL website to get yourself a screen name, you get to try using any screen name you like, but the AOL website will reject your choice if someone else is already using it. As a result, many people start by trying something like "tom" as a screen name and end up with "tomtom29384" by the time they find an alternative that someone else has not chosen before them.

Another approach is to give addresses a structure that makes it possible to determine that an address is unique by only looking at a small, local collection of other addresses. This is how street addresses work. If someone builds a new house on my street, they do not have to search the entire world to make sure the house's address is unique. My street is one block long. There are only four houses currently on the street. The existing houses are numbered 24, 30, 36, and 40. As long as the new house is given a number other than 24, 30, 36 or 40, its address is guaranteed to be unique. Even though my short street makes this approach particularly easy, it works for any street. As long as a new building's address is different from all of the other addresses on its street, then we know the address is unique throughout the entire world.

The fact that an address is unique does not imply you can easily find that address. The address 95 Taneytown Road happens to be a unique address. There is only one town in the world where one can find 95 Taneytown Road. Knowing this, however, does not tell you what town or even what country you should visit to find 95 Taneytown Road.[3]

[3]95 Taneytown Road is the address of a famous site in Gettysburg, Pennsylvania. It is, therefore, *a* Gettysburg

The structure of street addresses does a bit more than make it easy to ensure that new addresses are unique. It also provides a systematic way to find the house associated with an address. A street address is composed of a hierarchy of components that identify larger and larger geographical regions as one moves from the beginning of the address to the end. This becomes more obvious if one considers the components in reverse order. In my home address

40 Talcott Road
Williamstown, Massachusetts

"Massachusetts" identifies an entire state, "Williamstown" narrows the location down to about 50 square miles in the northwest corner of the state, Talcott Road limits it to one of four houses, and 40 finally identifies a single building.

Being hierarchical, however, does not guarantee that an address provides a systematic way to locate the object identified. Telephone numbers illustrate this. Phone numbers are hierarchical. In the United States and Canada, the first three digits are the area code, which identifies a large geographical region. The next three digits identify an "exchange", a switching office that serves a smaller region, and the last four digits identify a particular phone attached to that exchange. At least that is how things worked a few decades ago. Cell phones changed all that. Now, the area code and exchange tell you something about where the phone's owner originally purchased a cell phone, but nothing certain about the phone's current location.

7.5.2 802.xx Addresses

Before we can begin to build an internet, we must have several physical networks of computers that we would like to interconnect and the machines on these physical networks must have physical network addresses. Even after all of these networks are combined to form an internet, internet protocol messages will be sent encapsulated within physical network protocol packets. The addresses that determine where these packets will be delivered are physical network addresses, not internet addresses.

Many of the physical networks in use today are based on protocols standardized by the IEEE (Institute of Electrical and Electronics Engineers). These include the IEEE 802.3 Ethernet protocol and the IEEE 802.11 wireless protocols. Unsurprisingly, the good engineers at IEEE not only used very similar numbers to name their protocols. They also used basically the same format for the physical network addresses used in these protocols. As a result, an Ethernet address looks quite a bit like a wireless address or an FDDI network address.

Computer network addresses look a bit different from street addresses. A network address is just a sequence of binary digits. IEEE 802 physical addresses consist of 48 binary digits. For example, my machine's Ethernet address is

000000000001101101100011100100110001001000010100

Long, binary addresses are hard on the eyes. As a result, those who work with networks have developed a variety of shorthands for presenting network addresses. At the very least, it is common to break the sequence of bits into 8 bit bytes:

00000000 00011011 01100011 10010011 00010010 00010100

address. Of course, it is not *the* Gettysburg Address. Instead, it is the address where the Address was delivered.

For 802.xx addresses, groups of 4 binary digits are normally replaced by a single symbol using the following table of *hexadecimal* (base 16) equivalents.

binary	hex	binary	hex	binary	hex	binary	hex
0000	0	0001	1	0010	2	0011	3
0100	4	0101	5	0110	6	0111	7
1000	8	1001	9	1010	A	1011	B
1100	C	1101	D	1110	E	1111	F

Pairs of hexadecimal digits are then separated by colons. When applied to my machine's address, this leads to the more compact description

00:1b:63:93:12:14

Addresses are assigned to Ethernet cards, wireless cards, and other IEEE 802 network adapter cards by the manufacturer. The addresses are physically embedded in the electronics of the device when it ships from the factory. You do not have to tell your computer what its Ethernet address is. It knows. All it has to do is ask the Ethernet card installed. Note that the address is actually associated with the network interface hardware not the computer. If the wireless card in your laptop fails and is replaced when you send the machine in for repairs, you will have a new wireless address when the machine is returned.

The addresses used by the IEEE protocols have an internal structure used to ensure that all addresses are unique. The first 24 bits of each address is a sequence assigned to the manufacturer. The remaining 24 bits are chosen by the manufacturer in such a way that they are different from the last 24 bits of the address of every network interface unit previously produced by that manufacturer. This is more than enough to ensure that any two machines on the same Ethernet or wireless network will have distinct addresses.

There is one interesting sequence of bits that is never used as the address of a computer on an IEEE 802 network but that can be used as the destination address of a packet. This is the address containing all 1s. In the hexadecimal notation this address is written as

FF:FF:FF:FF:FF:FF

When a packet is sent with this address as its destination, the transmission is treated as a broadcast intended for all other machines on the same physical network. We have seen that all packets sent on an Ethernet or a wireless network are in some sense broadcast. Normally a machine ignores any packet it receives whose destination address is different from its own address. If a packet contains the destination address FF:FF:FF:FF:FF:FF, however, all machines that receive it will process it.

7.5.3 IP Addresses

The addresses used by the IP protocol consist of 32 0s and 1s. In particular, the IP address associated with the machine I am using at the moment is

10010111110010111010001001011000

When they are included in a network packet and transmitted through network wires, IP addresses must actually be encoded in binary form. They are, however, rarely presented in this way for human consumption. Instead, the are written using a system called *dotted decimal notation*. To convert an IP address into this form, you first break the sequence of binary digits that form the address into four 8-bit subsequences. For my machine's address the subsequences would be

10010111 11001011 10100010 01011000

Any sequence of binary digits can be interpreted as the description of a number in much the same way that we interpret numbers written using the 10 decimal digits. When the decimal digits are used to represent a number, the last digit represents 1s, the next to last digit represents 10s, the next to next to last digit represents 100s, and so on. Each digit correspond to a larger power of 10. Therefore, 2943 is interpreted as $2 \times 1000 + 9 \times 100 + 4 \times 10 + 3 \times 1$. To interpret a sequence of binary digits as a number we do the same thing with powers of 2 instead of powers of 10. Thus, the first subsequence in my machine's address, 10010111, would be interpreted as:

$$\begin{array}{llllllllll} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & = & \\ 1 \times 2^7 & + 0 \times 2^6 & + 0 \times 2^5 & + 1 \times 2^4 & + 0 \times 2^3 & + 1 \times 2^2 & + 1 \times 2^1 & + 1 \times 2^0 & = & \\ 1 \times 128 & + 0 \times 64 & + 0 \times 32 & + 1 \times 16 & + 0 \times 8 & + 1 \times 4 & + 1 \times 2 & + 1 \times 1 & = & 151 \end{array}$$

To obtain the dotted decimal representation of an IP address, we convert each of the four subsequences of binary digits in the address into decimal numbers and list them separated by periods. For my machine's address, this procedure results in the description

151.203.162.88

You have probably seen IP addresses displayed in this form before. In our discussion of IP, we will normally use this format to describe addresses. It is important to remember, however, that the addresses actually used in packets are not transmitted in this form. They are transmitted as sequences of 32 binary digits.

Like street addresses, IP addresses have a structure that both makes it easy to ensure that each address is unique and provides a systematic way to locate the computer associated with a specific address. Recall that the whole idea behind the internet concept is that a network can be built by interconnecting existing networks. The structure of an IP address reflects the structure of the Internet. One part of every IP address identifies one of the physical networks that is part of the Internet. The rest of the address identifies a particular machine within that physical network.

In street addresses, the most specific part of the address comes first, e.g., 40 Talcott Road, and the general location comes last, e.g., Williamstown, MA. In IP addresses, the order is reversed, the first bits of the address identify some physical network and the remaining bits identify a particular machine on that network.

We can use my machine's IP address, 151.203.162.88, as an example. The first 16 bits of this address, the bits 1001011111001011 corresponding to the prefix 151.203, are associated with a physical network operated by my Internet service provider, Verizon. This network is operated by Verizon to service DSL customers in my town and nearby towns. All machines connected to this network will have IP addresses that start with these bits. We can think of 151.203 as the address of this network.

Assigning addresses in this way makes it easy to ensure that my machine's address is unique. To obtain permission to use the prefix 151.203 as the address of one of its network, Verizon or any other organization that wants to operate a network connected to the Internet has to contact an organization named ICANN, the Internet Corporation for Assigned Names and Numbers. ICANN assigns numbers to networks in the same sense that AOL assigns IM screen names. Before it gives a company like Verizon permission to use 151.203, ICANN makes sure that no other organization has been given its permission to use this number.

Once Verizon knows that it has ICANN's blessing to use 151.203, it can assign addresses to the individual machines connected to its network without repeatedly consulting with ICANN. When Verizon assigns my computer the network address 151.203.162.88, all it has to do is check its own records to make sure that it has not told any other computer that connects to its network in my area to use the numbers 162.88 as the suffix of its network address.

This address structure also makes it easier to deliver packets within the Internet. When a machine looks at the destination address of a packet, it can divide the address up into its network part and the part that identifies a particular machine on that network. Until a packet reaches the network to which its destination is attached, the part of the address that identifies the particular machine is not very important. First, the routers that interconnect networks have to find a way to get the packet to that network. This can be done using just the prefix that identifies that network. Since there are far more machines in the Internet than networks, this makes the task the routers have to perform much simpler.

7.5.4 Classes, Subnets, CIDR, etc.

We have seen that there are 2^N distinct binary sequences composed of N binary digits. This implies that there are only $2^{32} = 4,294,967,296$ possible IP addresses. In the 1970s, when the Internet Protocol was designed, 4 billion addresses seemed like quite enough. When IP was first introduced, the Internet connected less than 1000 computers. By now, however, the fact that the number of possible Internet addresses is smaller than the population of the Earth is becoming a limitation.

In fact, the number of IP addresses that can be used in practice is far smaller than 4 billion. For example, Verizon associates the 16 bit address prefix 151.203 with the network through which my computer connects to the Internet from home. Since this prefix is 16 bits long and a complete IP address is 32 bit longs, there are $2^{16} = 65536$ address that start with this prefix. Only Verizon can use addresses starting with this prefix and it can only assign such addresses to computers attached to the same physical network as my computer.

My town is located in a relatively rural corner of Massachusetts. The population of our entire county, including such metropolitan centers as Pittsfield and Great Barrington, is roughly 140,000. Chances are that Verizon has less than 65,536 Internet service customers in this area. Suppose that Verizon only has 24,000 customers using the network in my region. In that case, 41,536 of the possible Internet addresses starting with 151.203 will not be used by Verizon and cannot be used anywhere else in the Internet. These addresses will be wasted.

If all IP addresses consisted of a 16 bit network prefix followed by a 16 bit machine number, so many addresses would be wasted that we would have run out years ago. Instead, a variety of techniques are available that make it possible to allocate ranges of addresses more flexibly.

First, it was never the case that all addresses were partitioned into two 16 bits subparts. From the very beginning, IP supported three main classes of addresses that used different schemes to partition the complete address into a network part and a machine part. The classes are named A, B, and C. In class B addresses, like the address Verizon associates with my computer, the first 16 bits are the network number while the last 16 bits are the machine number. Class C addresses are designed to support smaller networks more efficiently. The first 24 bits of a class C address identify a network leaving just 8 bits for the machine number. Class C addresses can support at most 256 machines. Finally, class A addresses use only the first 8 bits to identify the network. Class A addresses are intended for very large network. A class A network uses 24 bits for machine numbers allowing up to 16,777,216 distinct machine addresses.

Class	Binary Format	Dotted-decimal Format
A	0nnnnnnn mmmmmmmm mmmmmmmm mmmmmmmm	0 --- 127 . MMM . MMM . MMM
B	10nnnnnn nnnnnnnn mmmmmmmm mmmmmmmm	128 --- 191 . NNN . MMM . MMM
C	11nnnnnn nnnnnnnn nnnnnnnn mmmmmmmm	192 --- 255 . NNN . NNN . MMM

where:

n = 1 binary digit interpreted as part of a network number
m = 1 binary digit interpreted as part of a machine number
NNN = decimal number between 0 and 255 representing part of network number
MMM = decimal number between 0 and 255 representing part of machine number

Figure 7.18: IP address class formats

To process a packet, a computer on the Internet needs to be able to extract the network prefix from an IP address easily. This requires a way to quickly determine the class of an address. To facilitate this, the class of each address is encoded in the first two bits of the address. If an address starts with the binary digits 00 or 01, then it is a class A address, if it starts with 10 it is a class B address, and if it starts with 11 it is a class C address. Since addresses are usually written in dotted decimal rather than binary, it is helpful to know that this corresponds to treating dotted decimal addresses that start with numbers between 0 and 127 as class A addresses, addresses that start with numbers between 128 and 191 as class B addresses and addresses starting with 192 or greater as class C addresses.[4] Figure 7.18 illustrates the formats of class A, B and C addresses.

While these three address classes make it possible to allocate ranges of addresses more efficiently, they do not really provide much flexibility. One size fits all is clearly bad, but three sizes fit all isn't much better. As a result, other schemes have been developed for determining which bits of an IP address are interpreted as the network address and which bits identify a machine within a network. Two schemes you are likely to encounter are called *subnetting* and *CIDR* (pronounced like "cider" and short for "Classless Internet Domain Routing"). Both of these schemes make it possible to tell machines to partition an IP address into its network part and machine part in different ways.

With subnetting, each machine is given a string of 32 bits called a *subnet mask*. The bits in the mask are meant to correspond to the bits in an IP address. Each position in the mask with value 1 indicates that the corresponding bit from an IP address should be treated as part of the network number. Each position in the mask with value 0 indicates the corresponding bit from an IP address is part of the machine number. CIDR is a more obvious extension of the three class system. With CIDR each address must be accompanied by a number between 1 and 31 that indicates how big of a prefix of the address should be treated as the network part.

7.5.5 Domain Names

While you have probably seen IP addresses from time to time while using network software, they are not the first thing that comes to mind when you think of ways to identify machines on the Internet. When you want to read the news, you think of www.cnn.com or www.nytimes.com rather than 64.236.91.24 or 199.239.136.245, but somehow the messages that your computer has to send

[4]We are making a slight simplification here. There are two other special purpose address classes, D and E, that occupy part of what we have described as the class C range.

to request the news from CNN or the New York Times must include IP address like 64.236.91.24 and 199.239.136.245 to actually get the information you want.

Names like www.cnn.com and www.nytimes.com are called domain names. The Internet supports a mechanism called the Domain Name Service that enables computers to obtain a machine's IP address given its domain name. Computers called *domain name servers* are operated throughout the Internet to implement this mechanism. Each of these servers maintains a database listing the names and IP addresses of many machines and the addresses of servers that can be used to look up other names.

When you first use a name like www.cnn.com or www.gmail.com in a network application, your computer sends a request containing this name to a domain name server. The server returns an IP address associated with this name. Your machine uses this address to send your messages to the server.

To avoid making an excessive number of domain name service requests, machines typically save the answers they obtain from domain name servers for at least a few hours. If the same name is used again within that time, the computer will used the saved IP address rather than sending a new request to a domain name server. This collection of saved information is called the *domain name cache*.

Domain names are a convenience provided so that humans do not have to deal with ugly IP addresses. All of the fundamental mechanisms for sending messages through the Internet depend on IP addresses rather than domain names. When an application on your computer wants to send a message to a web server or a mail server, it needs to know the server's IP address. The Domain Name Service does not change this, but it enables your computer to hide these address from you by letting you to identify machines using names that are easier to remember than the corresponding addresses.

7.6 Who Am I?

In our discussion of hardware addresses, we explained that a machine can determine its own Ethernet or wireless network hardware address by simply asking its network interface card. These addresses are encoded directly in the hardware. IP addresses are not embedded in the hardware. They have to be assigned more flexibly than hardware addresses. Since every IP address on a given hardware network must start with a common prefix, the IP address associated with a machine must change when it is moved from one hardware network to another. Thus, every time a machine is turned on, rebooted, or just moved from one network to another, it needs some way to find out what IP address it is supposed to use.

There are many ways that a machine can be told what IP address to use. In the early days, IP addresses were assigned manually by typing an address provided by the network administrator into an appropriate file or dialog box. Most systems still support this approach. For example, Figure 7.19 shows the dialog box used in MacOS X to manually enter a machine's IP address (and several other pieces of network configuration information). By replacing the address "0.0.0.0" in the field labeled "IP Address" and clicking "Apply Now", a user could tell a machine to use any IP address. Of course, if the address started with the wrong prefix or was already being used by another computer, things might not work so well.

Since manual configuration of IP addresses can be awkward and error prone, most networks depend on protocols that provide an automatic way for computers to determine their IP addresses.

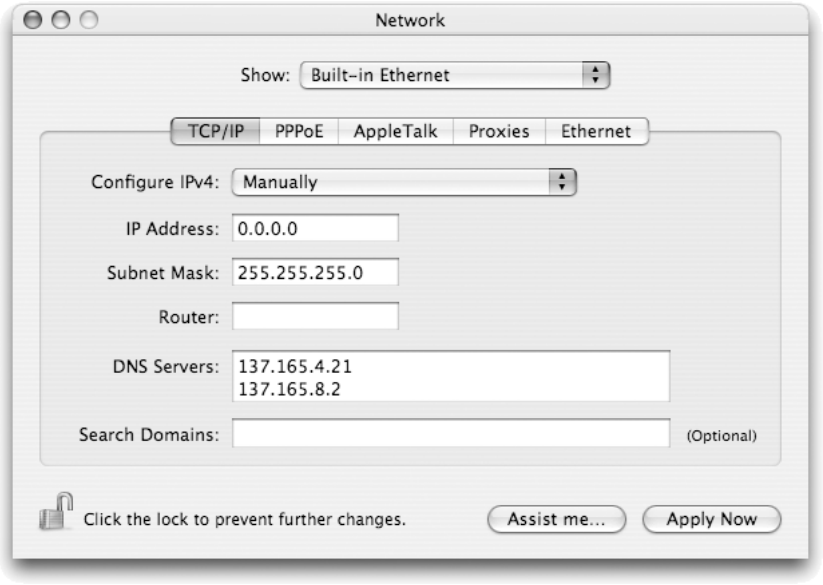

Figure 7.19: MacOS X dialog box for manual IP configuration

Several protocols have been used for this purpose including RARP (Reverse Address Resolution Protocol) and BOOTP (Bootstrap Protocol). Currently, the protocol that is most widely used to assign IP addresses is DHCP, the Dynamic Host Configuration Protocol.

When DHCP is used, a DHCP server must be installed on each physical network. This server will respond to a request from a machine that is new to the network by sending that machine the IP address it should use. On large networks, network administrators install, configure, and maintain a machine that acts as a DHCP server. On smaller networks, a DHCP server is often run on a machine providing several other services. In particular, on many wireless networks, the hub not only acts as a router. It also runs a DHCP server.

When a machine is first attached to a network, it does not know its own address, it does not know the IP address of the network to which it has been connected or the address of the machine that is acting as a DHCP server. Somehow, however, it has to send a message to the DCHP server asking for its help. It does this by using the broadcast support provided by many hardware networks. On an IEEE 802 network, the machine sends a DHCP packet requesting the IP address it should use to the broadcast address FF:FF:FF:FF:FF:FF. All machines on the network will therefore receive all DHCP requests. Most machines will be programmed to ignore such packets. Only the computer that acts as the DHCP server for the network will respond to the new machine.

The response a new machine receives from a DHCP server will actually contain more than just the IP address the machine should use. It will typically provide the same information that could have been manually provided through the dialog box shown in Figure 7.19. The most important items are the IP address the machine should use, the IP address of a router the new machine can use to send messages to destinations outside its hardware network, and the IP address of at least one domain name server.

There are several ways that a DHCP server can choose the IP address it assigns to a machine that sends it a request. The approach used depends on the priorities of the individual/organization

that created and maintains the network.

In cases where it is important to limit access to a network to authorized users, the DHCP server can be manually configured with a list of the hardware addresses of machines authorized to use the network and the IP address to be used by each of these authorized machines. When the server receives a DHCP request packet, it can extract the hardware address from the source address field of the request, look the address up in its table of authorized computers, and send the machine the assigned IP address.

In cases where limiting access is not so critical, a more dynamic approach can be used. The server is configured with a range of IP addresses it can assign. When it receives a request, it responds by sending the new machine any unused address in the allowed range. To avoid assigning the same address to multiple machines, the server must dynamically maintain a table of which addresses are currently in use. To make this possible, DHCP allows a server to place a time limit on the addresses it assigns. This time limit is called a *lease*. If a machine needs to continue using an address for longer than the specified time limit, it simply sends the server a request to renew its lease. If a computer leaves the network (or simply crashes) without telling the server, its address becomes usable again when the lease expires.

7.7 Packet Forwarding

An IP packet traveling through the Internet will usually be transmitted several times to traverse a multi-step path through the network. The first transmission will occur at the computer at which the packet was originally created. Unless the packet's destination is attached to the same network as its source, the first transmission will be addressed to a router that is connected to the same network as the source. This router will then transmit the packet a second time. Again, unless the destination is attached to the same network as the router, the next destination will be a second router. The process will repeat in this way until the packet arrives at a router connected to the same network as its ultimate destination. Then, the packet will be sent directly to its final destination.

Each of these transmissions will involve a hardware packet whose data field contains a copy of the original IP packet created by the source computer. The fields of this original IP packet will not be changed from one transmission to the next. In particular, the source and destination IP addresses will remain unchanged as the IP packet is encapsulated in one hardware packet after another.

The addresses in the hardware packets, on the other hand, will be different at each step in the process. While the IP packet is making a multi-step journey, each hardware packet in which it is encapsulated is traveling through just one network. The source address in each of these packets must be the address of the computer that actually transmitted the packet. That is, the source address used in the hardware packet for the first transmission must be the hardware address of the original source of the IP packet. This hardware address will refer to the same machine as the source address in the IP packet it encapsulates. The source address in the second hardware packet, on the other hand, will refer to the first router on the path. In general, the source address in the $n+1$st hardware packet used to encapsulate the IP packet will be the nth router in its path.

Similarly, the destination addresses used in each of the hardware packets that carry an IP packet will be different. The first hardware packet's destination address will be the address of the first router. The last hardware packet's destination address will be the address of the machine associated with the original IP packet's destination address.

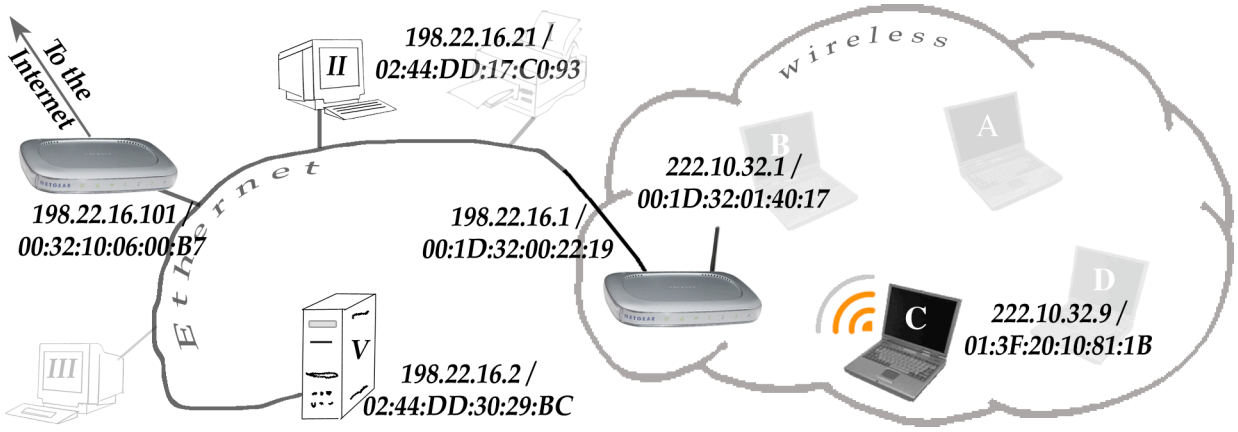

Figure 7.20: IP and hardware addresses for a small internet

In this section, we introduce the techniques used in the software that implements the Internet Protocol to determine what hardware addresses to place in the packets that carry IP packets toward their destinations. We will start in the next subsection by discussing the simple, special case of an IP packet that is sent to another machine attached to the same hardware network as the source. Using this example, we will introduce ARP, an important protocol that supports the implementation of IP. Then, we will consider the general problem of determining the addresses to use for packets that travel through multiple networks to reach their destinations.

The examples we use to illustrate how IP forwarding is implemented will all be based on the network fragment depicted in Figure 7.20. This figure shows the hypothetical network used as an example earlier in this chapter with three changes.

1. Network addresses are now associated with the machines that will play important roles in our examples,

2. the images of computers that will not be used in our following examples are grayed out, and

3. most importantly, one of the computers in the original diagram has been replaced by an additional router.

The router added in the figure is intended to represent a router through which the two networks shown in the figure can communicate with the rest of the Internet. Our example is no longer just an internet, it can now be thought of as part of THE Internet.

The hardware addresses shown in the diagram use the IEEE 802 format described above. The IP addresses shown in the diagram use two different prefixes, reflecting the fact that there are two distinct hardware networks in this Internet fragment. The Ethernet in the diagram is given the class C network address 198.22.16. The addresses associated with the two computers on this network and with the router connections to this network all start with this prefix. The wireless network prefix is 222.10.32. For the wireless router, we show addresses for both its connection to the Ethernet and its connection to the wireless network. For the wired router that connects this fragment to the rest of the Internet, we only show the address of its connection to the Ethernet in our diagram. We don't make any assumptions about what other type or types of networks it is connected to other than that they provide a path to the rest of the Internet.

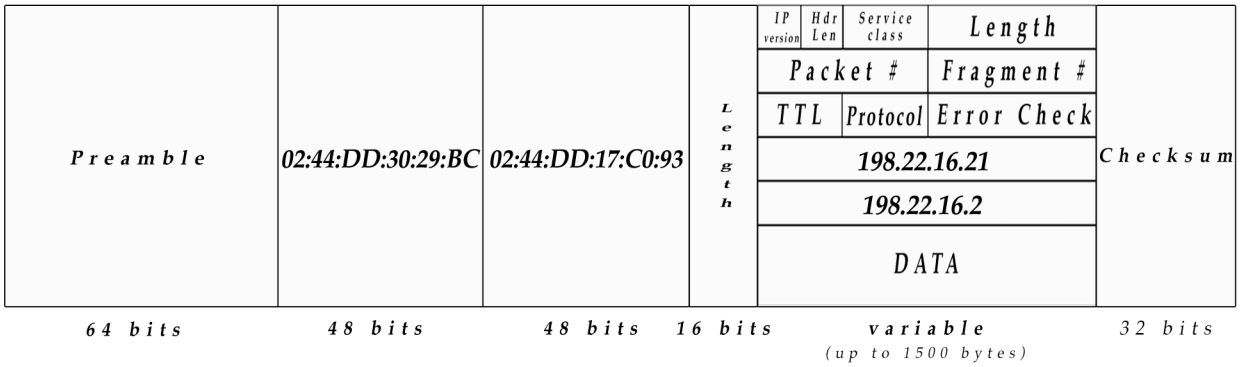

Figure 7.21: IP packet from computer II encapsulated for delivery to V

7.7.1 Address Resolution

Suppose that the computer identified as "II" in Figure 7.20 wants to send a message to the computer labeled "V". Since this message only needs to travel between computers on a single hardware network, it will not really take advantage of the power of the Internet protocol. Part of providing the illusion of an internet, however, is making it unnecessary for application programs to notice that a message like this is a simple, special case. Therefore, the program that creates this message will still encode it using the IP packet format and pass it on to the operating system for transmission. The operating system will then encapsulate the IP packet within an Ethernet frame including the correct addresses for transmission.

The addressing details of the Ethernet frame that would be transmitted for such a message are shown in Figure 7.21. The IP packet created by the application program is embedded within the data field of this Ethernet frame. The rest of the fields in the Ethernet frame must be produced by code in the operating system. In particular, the operating system must determine the Ethernet source and destination addresses.

Filling in the Ethernet source address is easy. As mentioned earlier, Ethernet addresses are encoded in the hardware of Ethernet adapters. The operating system can therefore simply ask the machine's Ethernet adapter to tell it what its Ethernet address is.

Filling in the Ethernet destination address is not as easy. The operating system needs some way to take the IP destination address provided by the application program and find the Ethernet address of the same machine. Finding the Ethernet address associated with an arbitrary IP address could be very difficult. Recall, however, that we are assuming that the destination is on the same hardware network as the source. To handle IP packets like this, the operating system just needs a way to find the hardware addresses associated with IP addresses on the hardware network to which it is attached.

There are many ways to enable the operating system to find the hardware address of another computer given the other machine's Internet address. In the early days of the Internet, each machine contained a file listing the IP and Ethernet address of all computers installed on its network. The operating system would simply search this file. Such a scheme, however, is difficult to maintain.

The files of addresses would have to be updated manually on every machine whenever a machine was added to the network.

As a result, most modern Ethernets and wireless networks depend on a protocol called ARP (the Address Resolution Protocol) to solve this problem. Like DHCP, ARP takes advantage of the broadcast nature of Ethernets and wireless networks. A computer that wants to determine the hardware address associated with some IP address places the IP address within an ARP packet and broadcasts the packet on its network. Machines that receive such an ARP packet interpret it as a request for the hardware address associated with the IP address found in the packet. If the IP address is valid, there will be at least one machine that can answer this question, the same machine that is associated with the IP address. The rules of the ARP protocol call for that machine to send an ARP packet containing its hardware address back to the machine that made the request.

ARP has one feature in common with the Domain Name Service. After a machine receives the answer to an ARP request, it saves the answer for a reasonable amount of time in a table called the ARP cache. Before sending an ARP request, each machine checks its own ARP cache to see if the cache can provide the needed hardware address. If so, no ARP request is sent. This cache ensures that in most cases only the first attempt to communicate with another machine requires an ARP packet.

Both ARP and the Domain Name Service translate names/addresses of one form into addresses of another form. They also both use caches to reduce the overhead of such address requests. These two systems, however, differ in two important ways. First, ARP can only be used to determine the addresses of machines attached to the same hardware network as the machine that initiates the request. The Domain Name Service can be used to look up any name in the Internet. Second, the Domain Name Service depends on dedicated servers to respond to requests. There is, on the other hand, no such thing as an ARP server. ARP depends on the cooperation of all machines on a network to respond to requests for their own addresses. It is a fully distributed lookup mechanism.

7.7.2 One Step at a Time

In the preceding section, we restricted our attention to the simplest type of IP packet to deliver, a packet sent between two computers on the same hardware network. In this section, we will look at the general case of how to forward a packet whether its final destination is on a local network or on a network that can only be reached by traveling through a dozen or more routers.

One surprising fact is that the same algorithm can be used to determine the address to which a packet should be forwarded in both computers acting as routers and computers with single connections to the network. A machine applies this algorithm whenever it receives a new IP packet. In a client machine, the source of the packet to transmit is usually some application running on the machine that has created the packet and passed it to the operating system for transmission. On a router, the source of the packet is usually the network itself. Either way, the software in the machine/router has to determine how to handle the packet based on the IP destination address found in the packet. As a result, the best way to describe the algorithm is to ignore where the packet actually came from. That is, the algorithm we provide will describe how all machines should handle packets whether those packets are created by applications running on the machine or received through one of the machine's connection to the network.

The structure of IP addresses is critical to the IP software's ability to forward packets correctly. Recall that each IP address can be divided into one part that identifies a specific hardware network within the Internet and a second part that identifies a particular machine on that hardware network.

The algorithm for forwarding packets depends on this property of IP addresses to determine whether or not a packet can be delivered directly.

A machine can determine whether a packet can be delivered directly by comparing the network part of the packet's IP destination address to the network part of the machine's own IP addresses. Note that we deliberately talk about address**es** rather than a single address. In this way, we accommodate both routers and other machines connected to the network. A router will have several connections to the network and a distinct IP address with a distinct network prefix for each of these connections. To decide whether a packet can be delivered directly to its destination through any of the networks to which it is connected, a router must compare all of its addresses to the packet's destination address.

The complete algorithm used to determine how to handle a new IP packet consists of three steps or cases. We will first describe all three cases very briefly. We will then explore each case in more details using examples to illustrate when and how it would apply.

The IP Packet Forwarding Algorithm

Case 1 – The packet's destination IP address is equal to one of this machine's IP addresses.

In this case, the machine that is executing the algorithm is the packet's final destination. The packet should be delivered to the appropriate application software on the machine.

Case 2 – The network prefix of the packet's destination IP address is equal to the network prefix of one of this machine's IP addresses.

In this case, the packet should be sent through the network associated with the matching network prefix directly to the destination machine. The ARP protocol will typically be used to determine the correct hardware address associated with the IP destination address found in the packet. This hardware address will then be used as the destination address in the packet that encapsulates the IP packet as it travels to its destination.

Case 3 – The network prefix of the packet's destination IP address does not match any of the network prefixes of this machine's IP addresses.

In this case, the packet must be sent to an appropriate router for further forwarding. The IP address of the router that should be used will be determined using default router information and/or routing table information as described below. Once the correct router's IP address has been determined, the ARP protocol will typically be used to determine the hardware address associated with the router's IP address. This hardware address will then be used as the destination address in the packet that encapsulates the IP packet as it travels to its destination.

The first case in this algorithm is a bit odd. It is the "don't forward" case of the "forwarding" algorithm. It is present because we want to view this algorithm as the procedure *all* machines use to process *all* packets. When a packet arrives at its final destination, we do not want to forward the packet to some other machine. We want to deliver it to the appropriate software module on the destination machine.

The second case of the algorithm handles the important case we described in Section 7.7.1 where a machine is sending a packet to another machine attached to the same hardware network. In the example presented in Section 7.7.1, the packet in question traveled directly from its source to its destination. This component of the algorithm, however, also handles situations where a packet is being sent from the last router in a multi-step path to its destination.

As an example, suppose that the laptop labeled "C" in Figure 7.20 sent an IP packet to the machine labeled "V" on the Ethernet in the figure. The IP destination address in this packet will be 198.22.16.2. Following case 3 of our algorithm, computer "C" will encapsulate the IP packet in a wireless packet and send it to the router connecting the wireless network and the Ethernet shown in the figure. At this point, the router will have to process the packet to determine its next step. The router will compare the packet's destination address, 198.22.16.2, to both of the addresses associated with the router, 222.10.32.1 and 198.22.16.1. It will notice that the network prefix of the destination address matches the network prefix of the router's connection to the Ethernet. It will therefore apply case 2 of the algorithm and send the IP packet directly to its destination encapsulated in an Ethernet packet.

In all other situations, case 3 will apply and the packet will be forwarded to a router for further processing. In some situations, it is very easy to determine which router should be used. For example, there is only one router attached to the wireless network, the router with IP address 222.10.32.1. Any packet that cannot be delivered locally on this network must be forwarded to this router. In such a situation, all the machine's in the network need just one piece of information to apply case 3 correctly. They need to be told the IP address of the "default router."

We briefly discussed how machines are told the address of their default router in Section 7.6. This information is usually included in the response a machine receives when it makes a DHCP request to obtain its own IP address. Note that the default router is described using its IP address rather than its hardware address. When actually sending a packet to the router a machine will use ARP to convert the router's IP address into a hardware address. Since many packets will probably be sent through the router, the hardware address for the router will typically be available in the ARP cache.

Things get a bit more complicated if there is more than one router attached to a machine's network. Suppose that after receiving a message from laptop "C", machine "V" tries to send a response back to "C" in an IP packet. The destination address for this packet will be 222.10.32.9. When "V" compares this address to its own IP address, 198.22.16.2, it will realize that it must forward the packet to a router following case 3. Unfortunately, there are two routers attached to the same network as "V". These routers have the IP addresses 198.22.16.101 and 198.22.16.1. How should "V" decide which router to use?

To enable machines to select an appropriate router, the IP software maintains a collection of information called a *routing table* or *forwarding table*. Each entry in the table consists of the network prefix for some hardware network and the address of the best router to use as the first step to get to that network.

The routing table can be understood as a list of exceptions to the use of the machine's default router. That is, if a machine like "V" wants to send a packet to network 222.10.32, it first looks in the routing table to see if it has some specific information about what router to use to reach 222.10.32. If so, it sends the packet to the router identified in the table. If not, it sends the packet to its default router. In particular, for the network shown in Figure 7.20, the routing table for "V" and all other machines on the Ethernet should contain just one entry:

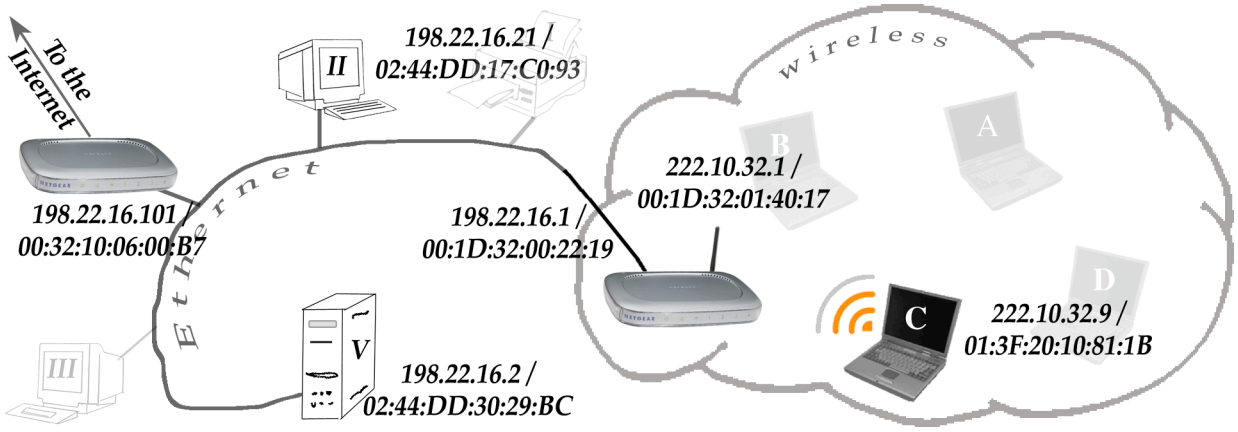

Figure 7.22: IP and hardware addresses for a small internet

222.10.32 → 198.22.16.1

All machines on the Ethernet should also be told to use 198.22.16.101 as their default router. Given this information, packets destined for machines on the wireless network will be sent to 198.22.16.1 while all other packets will be sent to 198.22.16.101. In particular, a packet destined for laptop "C" would be sent to 198.22.16.1.

While a very simple routing table suffices for machines on the Ethernet shown in Figure 7.20, it should be clear that a machine on a network in some location central to the Internet might require a very large collection of entries in its routing table. It is not clear how the information required to construct such a table should be collected. In fact, machines on the Internet depend on complicated routing protocols to collect and distribute the information required to construct routing tables. Two important examples of such routing information protocols are BGP (the Border Gateway Protocol) and OSPF (Open Shortest Path First). We will discuss some of the algorithms used in these protocols in Chapter 8. For now, we will just assume this information will be available as required to execute step 3 of our forwarding algorithm.

7.8 From Start to Google

We have now explored all of the fundamental mechanisms that make it possible for machines to deliver messages through the Internet. There are clearly many ideas and mechanisms to understand from the format of IP addresses to the behavior of a DHCP server to the important notion of encapsulation. In the preceding sections, we have examined each of these topics separately. In this section, we will both summarize and synthesize what we have presented by briefly explaining the key networking events that occur when one turns a computer on and uses it to contact www.google.com.

To make this example concrete, we will assume the computer that is being used is the laptop labeled "C" in our hypothetical network. To spare the reader from unnecessary page flipping, the diagram of this network is repeated in Figure 7.22.

As soon as computer C is turned on, its operating system software is loaded into the machine and starts running. The operating system does many things during startup, most of which have little to do with the network. When it gets around to worrying about the network, it first checks the

hardware network interfaces to see what, if any, network the machine can use for communications. A laptop often has both an Ethernet jack and a wireless card. The system checks to see whether a cable carrying an active signal is plugged into the Ethernet jack or whether it can join a nearby wireless network. If both are available, a user-controlled system preference determines which network is used as the default. In the case of computer "C", it will detect and decide to use the wireless network.

Even after it knows which network it will use, the machine still does not know how to use that network for IP communications. In particular, it does not know the IP address to use for itself or for a router. This is where DHCP comes in. The machine's operating system will broadcast a DHCP packet to all computer's within the wireless network.

Typically, the DHCP server for a wireless network runs on the same computer that acts as the hub/router for the network. Assuming that this is the case in Figure 7.22, the hub will receive the DHCP request from "C" and respond with a DHCP packet telling "C":

- to use 22.10.32.9 as its IP address
- to use the hub's address, 222.10.32.1, as its default router address

The DHCP server will also give "C" the address of a Domain Name Server. Unlike DHCP servers, it is not necessary to have a Domain Name Server on every network. In fact, all of the machines in Figure 7.22 might depend on one or more domain name servers that were located outside of the two networks shown. They would just send domain name lookup request through router 198.22.16.101 to reach such servers.

To keep things a bit simpler, we will assume that the organization that runs these two network has set up a domain name server running on the machine labeled "V. Therefore, the third piece of information "C" will receive in the response to its DHCP request will be the address to use for name server requests, 198.22.16.2.

Once the operating system finishes starting up, we assume that the user of the computer launches a web browser and tries to look something up on www.google.com either by selecting a Google bookmark or entering a search term in the browser's Google lookup field. Either way, the browser will then try to send a message to www.google.com.

To send a message to www.google.com, the browser needs the IP address for Google. This means that before it can attempt to send a packet to Google, it must send a Domain Name Service request to 198.22.16.2. It will put the appropriate data (including the name "www.google.com") into an IP packet and pass this packet to the operating system for transmission.

The operating system will now apply the IP packet forwarding algorithm to this IP packet. Comparing the address of the name server, 198.22.16.2, to the machine's own address, 22.10.32.9, it will realize that the packet needs to be sent to a router following case 3 of the forwarding algorithm. Therefore, the operating system will want to send the packet to its default router, 222.10.32.1. To do this, it has to encapsulate the packet for transmission within a wireless packet. This requires using the hardware address for the router as the destination address in the packet. To obtain this hardware address, the operating system on "C" will broadcast an ARP packet to all machines on the wireless network asking for information about 222.10.32.1. The hub will recognize this as a request for its own address and send an ARP response including its hardware address, 00:1D:32:01:40:17. The operating system will then send the IP packet containing the domain name lookup request to 00:1D:32:01:40:17. It will also save the fact that the hardware address associated with 222.10.32.1 is 00:1D:32:01:40:17 in its ARP cache.

When this packet reaches the wireless hub, the software in the hub will extract the IP packet from its wireless packet envelope and apply the IP forwarding algorithm to it. The hub will compare both of its IP addresses, 222.10.32.1 and 198.22.16.1 to 198.22.16.2, the destination IP address found in the IP packet containing the lookup request. It will notice that the network prefix 198.22.16 appears in both the packet's destination address and the address of its connection to the Ethernet. Therefore, it will follow case 2 of the forwarding algorithm and send the packet directly through the Ethernet to the machine labeled "V".

Of course, to send a packet directly through the Ethernet, the wireless hub has to obtain the Ethernet address associated with 198.22.16.2. It relies on ARP to obtain such information, but in this situation, it will probably not have to send any ARP packets. If 198.22.16.2 is the default name server used by machines on the wireless network, then unless "C" is the first machine to start up on that network, some other request destined for 198.22.16.2 has probably recently passed through the wireless hub. In this case, the fact that the Ethernet address for 198.22.16.2 is 02:44:DD:30:29:BC will be available in the hub's ARP cache. It will use this information to send the lookup request packet without first sending an ARP request.

When the Ethernet packet reaches 198.22.16.2, its operating system will extract the IP packet from the Ethernet envelope, recognize it as a domain name lookup request, and pass it on to the domain name server application running on the machine. By some magic that we will not explore, this application will determine that the IP address 72.14.205.103 is associated with Google. It will put this information in an appropriate IP packet and pass it back to the operating system to deliver to the machine specified as the source of the IP packet that contained the lookup request, 222.10.32.9.

The delivery of this response packet will follow the same pattern as the delivery of the request in reverse. When the operating system on the name server applies the forwarding algorithm to the response packet, it will realize it should forward it to a router following case 3 of the forwarding algorithm. All machines on the Ethernet will have been told to use 198.22.16.101 as their default router, but they will also have a routing table containing the entry

$$222.10.32 \rightarrow 198.22.16.1$$

Since the prefix of the address of this packet's destination matches 222.10.32, the router identified in the routing table entry will be used rather than the default router. That is, machine "V" will forward the response packet to the wireless hub. Most likely, "V" has exchanged other packets with the hub recently and will therefore have the hub's Ethernet address in its ARP cache. As a result, this packet can be sent without transmitting any ARP lookup requests.

Next, the wireless hub will realize it can deliver the response packet directly to C at address 222.10.32.9. Even though this will be the first IP packet it sends to C, the hub will also almost certainly have C's wireless address, 01:3F:20:10:81:1B, stored in its ARP cache. This is because of a little trick typically included in the implementation of ARP. When a machine receives an APR request looking for its address, it knows that it will probably receive an IP packet from that machine shortly afterwards. Most IP packets require responses. The machine can safely assume that it will soon have to send a packet to the machine that sent the ARP request. Therefore, in addition to responding to the request, a machine will add the IP and hardware addresses of the machine that sent the request to its ARP cache. As a result, the hub will already know C's wireless address and can forward the response to the domain name lookup request to C without sending another ARP request packet.

Whew! After all that work, "C" has finally received the information it needs to do what it really wants to do. It now has an IP address it can use to send a request to Google. The operating system on "C" will save this information in its domain name cache. It will also deliver the information to the web browser, the application that sent the name server request.

Now that the web browser knows Google's IP address, it can ask its operating system to send a request to the web server on Google at 72.14.205.103. C's operating system will apply case 3 of the forwarding algorithm and send this IP packet within a wireless packet to the wireless hub. The ARP cache should provide the needed hardware address for the hub.

Unlike the domain name lookup packet, the wireless hub will not be able to deliver this request directly. Google is too far away. Instead it will recognize that it has to apply case 3 of the forwarding algorithm and send the packet to a hub on the network. Like other machines on the Ethernet, the wireless hub will be configured to use 198.22.16.101 as its default router. Therefore, it will forward the packet to 198.22.16.101, the router that connects our two sample networks to the rest of the Internet.

We will refrain from considering the details of what happens between the point where this packet reaches 198.22.16.101 and when it arrives at Google. The overall process, however, will resemble what we have been describing. One router after another will forward the packet until it reaches Google's network. When it reaches Google's web server, it will be delivered to a server application that performs the requested Google search and sends back a list of hits in one or more IP packets. These packets will follow a similar path back to 198.22.16.101. From there, they will be forwarded to the wireless hub and then delivered directly to "C".

This, amazingly, is how the Internet works. To both users and application programs, the details of ARP lookups and the forwarding of encapsulated packets become invisible, yielding the impression of a single, unified network capable of delivering messages anywhere in the world. This illusion, however, depends on the interactions between these underlying components.