CHAPTER

Protocols and Layers

1

What You Will Learn

In this chapter, you will learn about the protocol stack used on the global public Internet and how these protocols have been evolving in today's world. We'll review some key basic definitions and see the network used to illustrate all of the examples in this book, as well as the packet content, the role that hosts and routers play on the network, and how graphic user and command line interfaces (GUI and CLI, respectively) both are used to interact with devices.

You will learn about standards organizations and the development of TCP/IP RFCs. We'll cover encapsulation and how TCP/IP layers interact on a network.

This book is about what actually happens on a real network running the protocols and applications used on the Internet today. We'll be looking at the entire network—everything from the application level down to where the bits emerge from the local device and race across the Internet. A great deal of the discussion will revolve around the TCP/IP protocol suite, the protocols on which the Internet is built. The network that will run these protocols is shown in Figure 1.1.

Like most authors, I'll use TCP/IP as shorthand for the entire Internet protocol stack, but you should always be aware that the suite consists of many protocols, not just TCP and IP. The protocols in use are constantly growing and evolving as the Internet adapts to new challenges and applications. In the past few years, four trends have become clear in the protocol evolution:

Increased use of multimedia—The original Internet was not designed with proper quality of service assurances to support digital voice and video. However, the Internet now carries this as well as bulk and interactive data. (In this book, "data" means non-voice and non-video applications.) In the future, all forms of information should be able to use the Internet as an interactive distribution medium without major quality concerns.

Increasing bandwidth and mobility—The trend is toward higher bandwidth (capacity), even for mobile users. New wireless technologies seem to promise

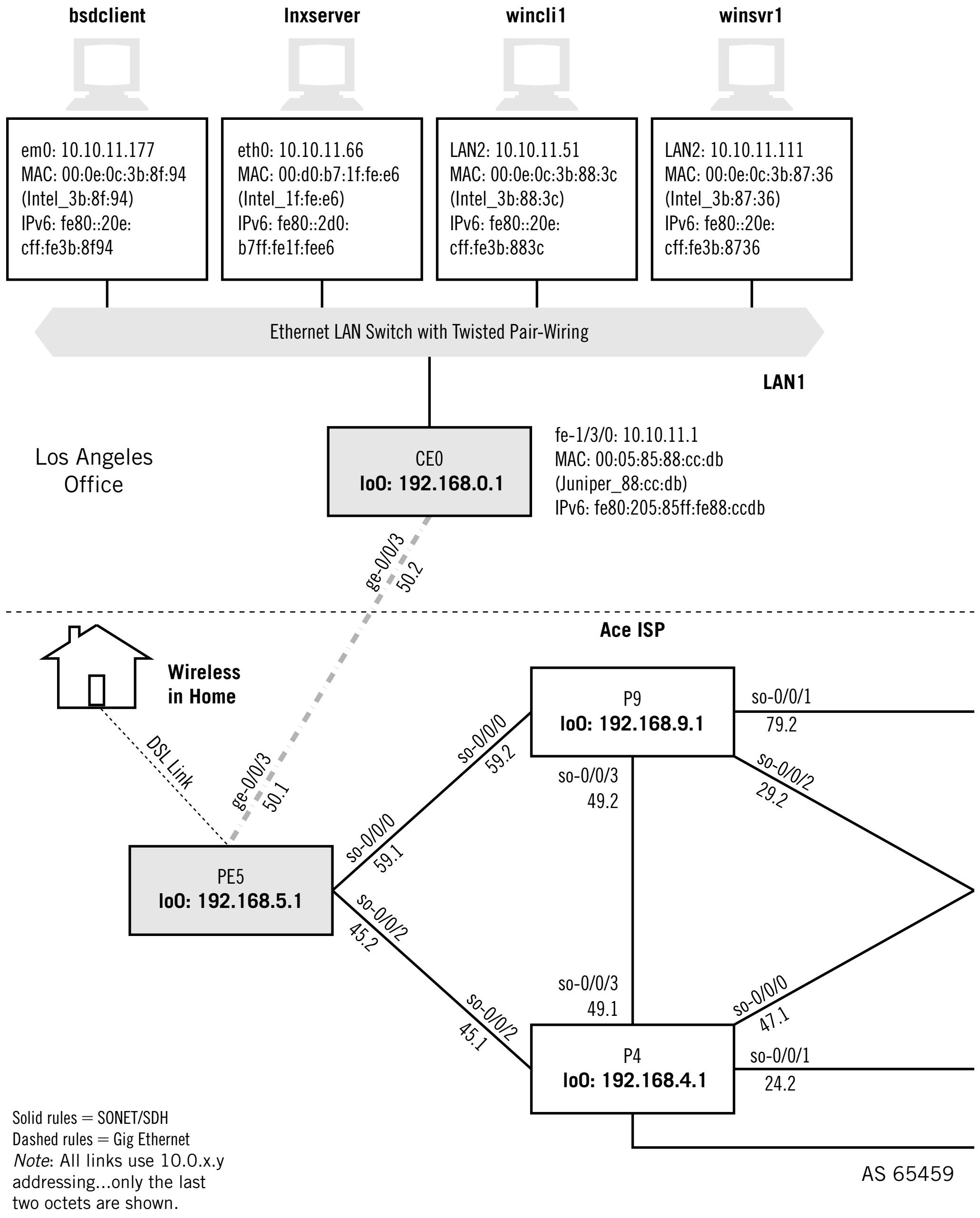
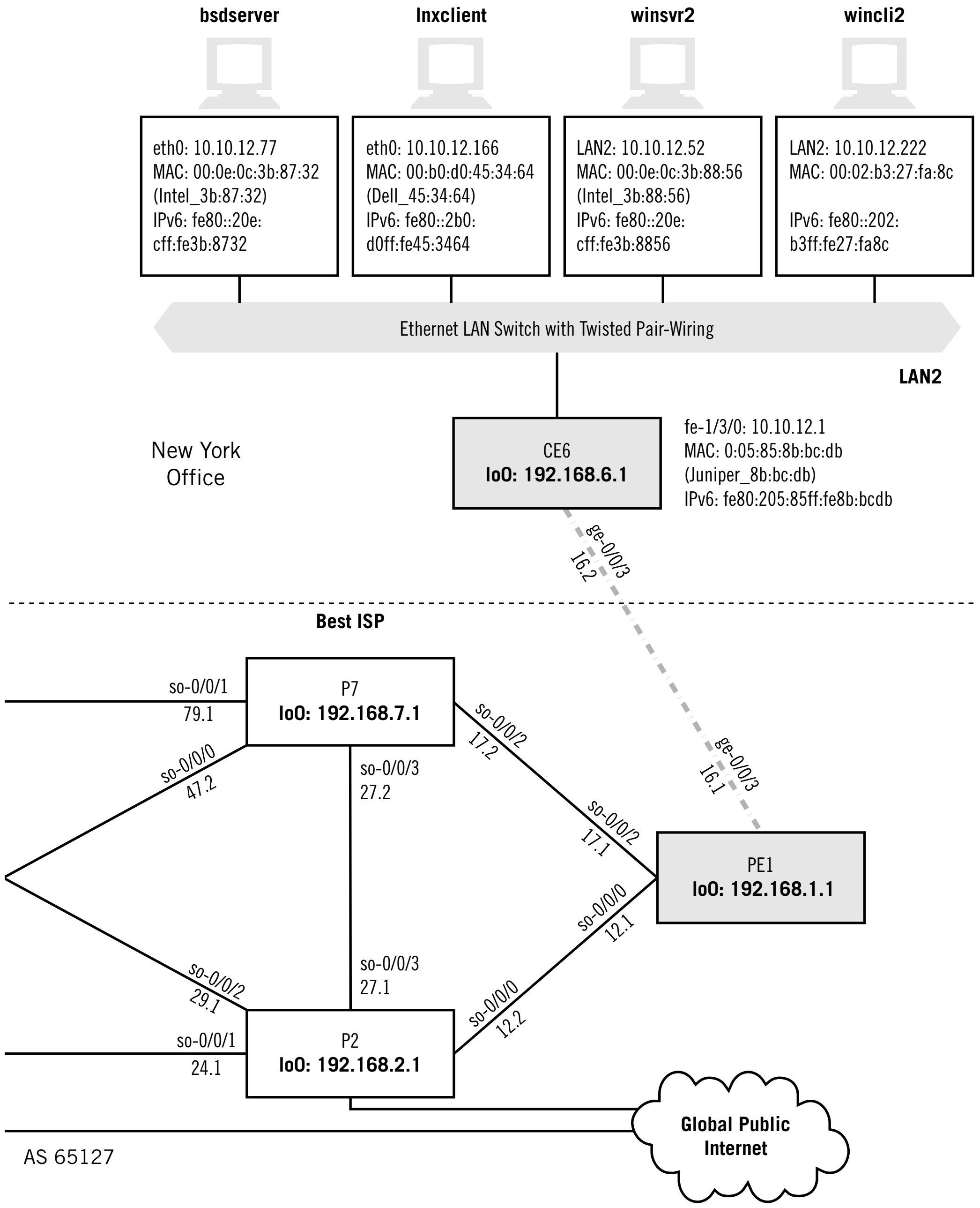


FIGURE 1.1

The Illustrated Network, showing the routers, links, and hosts on the network. Many of the layer addresses used in this book appear in the figure as well.


bsdserver
lnxclient
winsvr2
wincli2
eth0: 10.10.12.77
MAC: 00:0e:0c:3b:87:32
(Intel_3b:87:32)
IPv6: fe80::20e:
cff:fe3b:8732
eth0: 10.10.12.166
MAC: 00:b0:d0:45:34:64
(Dell_45:34:64)
IPv6: fe80::2b0:
d0ff:fe45:3464
LAN2: 10.10.12.52
MAC: 00:0e:0c:3b:88:56
(Intel_3b:88:56)
IPv6: fe80::20e:
cff:fe3b:8856
LAN2: 10.10.12.222
MAC: 00:02:b3:27:fa:8c
IPv6: fe80::202:
b3ff:fe27:fa8c
Ethernet LAN Switch with Twisted Pair-Wiring
LAN2
New York Office
CE6
lo0: 192.168.6.1
fe-1/3/0: 10.10.12.1
MAC: 0:05:85:8b:bc:db
(Juniper_8b:bc:db)
IPv6: fe80:205:85ff:fe8b:bcdb
ge-0/0/3
16.2
Best ISP
so-0/0/1
79.1
P7
lo0: 192.168.7.1
so-0/0/2
17.2
ge-0/0/3
16.1
so-0/0/0
47.2
so-0/0/3
27.2
so-0/0/2
17.1
PE1
lo0: 192.168.1.1
so-0/0/0
12.1
so-0/0/2
29.1
so-0/0/3
27.1
so-0/0/0
12.2
so-0/0/1
24.1
P2
lo0: 192.168.2.1
Global Public Internet
AS 65127

the "Internet everywhere." Users are no longer as restricted to analog telephone network modem bit rates, and new end-electronics, last-mile technologies, and improved wiring and backbones are the reason.

Security—Attacks have become much more sophisticated as well. The use of privacy tools such as encryption and digital signatures are no longer an option, but a necessity. E-commerce is a bigger and bigger business every year, and on-line banking, stock transactions, and other financial manipulations make strong security technologies essential. Identity verification is another place where new applications employ strong encryption for security purposes.

New protocols—Even the protocols that make up the TCP/IP protocol suite change and evolve. Protocols age and become obsolete, and make way for newer ways of doing things. IPv6, the eventual successor for IPv4, is showing up on networks around the world, especially in applications where the supply of IPv4 addresses is inadequate (such as cell phones). In every case, each chapter attempts to be as up-to-date and forward-looking as possible in its particular area.

We will talk about these trends and more in later chapters in this book. For now, let's take a good look at the network that will be illustrated in the rest of this book.

Key Definitions

Any book about computers and networking uses terminology with few firm definitions and rules of usage. So here are some key terms that are used over and over throughout this book. Keep in mind that these terms may have varying interpretations, but are defined according to the conventions used in this book.

- **Host:** For the purposes of this book, a host is any endpoint or end system device that runs TCP/IP. In most cases, these devices are ordinary desktop and laptop computers. However, in some cases hosts can be cell phones, handheld personal digital assistants (PDAs), and so on. In the past, TCP/IP has been made to run on toasters, coffee machines, and other exotic devices, mainly to prove a point.
- **Intermediate system:** Hosts that do not communicate directly pass information through one or more intermediate systems. Intermediate systems are often generically called "network nodes" or just "nodes." Specific devices are labeled "routers," "bridges," or "switches," depending on their precise roles in the network. The intermediate nodes on the Illustrated Network are routers with some switching capabilities.
- **System:** This is just shorthand for saying the device can be a host, router, switch, node, or almost anything else on a network. Where clarity is important, we'll always specify "end system" or "intermediate system."

THE ILLUSTRATED NETWORK

Each chapter in this book will begin with a look at how the protocol or chapter contents function on a real network. The Illustrated Network, built in the Tech Pubs department of Juniper Networks, Inc., in Sunnyvale, California, is shown in Figure 1.1.

The network consists of systems running three different operating systems (Windows XP, Linux, and FreeBSD Unix) connected to Ethernet local area networks (LANs). These systems are deployed in pairs, as either *clients* (for now, defined as "systems with users doing work in front of them") and *servers* (for now, defined as "systems with administrators, and usually intended only for remote use"). When we define the *client* and *server* terms more precisely, we'll see that the host's role at the protocol level depends on which host initiates the connection or interaction. The hosts can be considered to be part of a corporate network with offices in New York and Los Angeles.

Addressing information is shown for each host, router, and link between devices. We'll talk about all of these addresses in detail later, and why the hosts in particular have several addresses in varying formats. (For example, the hosts only have *link-local* IPv6 address, and not global ones.)

The LANs are attached to Juniper Networks' *routers* (also called intermediate nodes, although some are technically *gateways*), which in turn are connected in our network to other routers by point-to-point synchronous optical network (SONET) links, a type of wide area network (WAN) link. Other types of links, such as asynchronous transfer mode (ATM) or Ethernet, can be used to connect widely separated routers, but SONET links are very common in a telecommunications context. There is a link to the global Internet and to a home-based wireless LAN as well. The home office link uses digital

Major Parts of the Illustrated Network

The Illustrated Network is composed of four major components. At the top are two Ethernet LANs with the hosts of our fictional organization, one in New York and one in Los Angeles. The offices have different ISPs (a common enough situation), and the site routers link to Ace ISP on the West Coast and Best ISP on the East Coast with Gigabit Ethernet links (more on links in the next chapter). The two ISPs link to each other directly and also link to the "global public Internet." Just what this is will be discussed once we start looking at the routers themselves.

One employee of this organization (the author) is shown linking a home wireless network to the West Coast ISP with a high-speed ("broadband") digital subscriber line (DSL) link. The rest of the links are high-speed WAN links and two Gigabit Ethernet (GE) links. (It's becoming more common to use GE links across longer distances, but this network employs other WAN technologies.)

The Illustrated Network is representative of many LANs, ISPs, and users around the world.

subscriber line (DSL), a form of dedicated broadband Internet access, and not dial-up modem connectivity.

This network will be used throughout this book to illustrate how the different TCP/IP protocols running on hosts and routed networks combine to form the Internet. Some protocols will be examined from the perspective of the hosts and LAN (on the local "user edge") and others will be explored from the perspective of the service provider (on the global "network edge"). Taken together, these viewpoints will allow us to see exactly how the network works, inside and out.

Let's explore the Illustrated Network a little, from the user edge, just to demonstrate the conventions that will be used at the beginning of each chapter in this book.

Remote Access to Network Devices

We can use a host (client or server system running TCP/IP) to remotely access another device on the local network. In the context of this book, a *host* is a client or server system. We can loosely (some would say *very* loosely) define *clients* as typically the PCs on which users are doing work, and that's how we'll use the term for now. On the other hand, *servers* (again loosely) are devices that usually have administrators tending them. Servers are often gathered in special equipment racks in rooms with restricted access (the "server room"), although print servers are usually not. We'll be more precise about the differences between clients and servers as the "initiating protocol" later in this book.

Let's use host `lnxclient` to remotely access the host `bsdserver` on one of the LANs. We'll use the secure shell application, `ssh`, for remote access and log in (the `-l` option) as `remote-user`. There are other remote access applications, but in this book we'll use `ssh`. We'll use the command-line interface (CLI) on the Linux host to do so.

```
[root@lnxclient admin]# ssh -l remote-user@bsdserver
Password:
Last login: Sun Mar 17 16:12:54 2008 from securepptp086.s
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
The Regents of the University of California. All rights reserved.
FreeBSD 4.10-RELEASE (GENERIC) #0: Tue May 25 22:47:12 GMT 2004
Welcome to FreeBSD!...
```

We can also use a host to access a *router* on the network. As mentioned earlier, a *router* is a type of intermediate system (or network node) that forwards IP data units along until they reach their destination. A router that connects a LAN to an Internet link is technically a *gateway*. We'll be more precise about these terms and functions in later chapters dealing with routers and routing specifically.

Let's use host `bsdclient` to remotely access the router on the network that is directly attached to the LAN, router `CE0` ("Customer Edge router #10"). Usually, we'd do this to configure the router using the CLI. As before, we'll use the secure shell application, `ssh`, for remote access and log in as `remote-user`. We'll again use the CLI on the Unix host to do so.

```
bsdclient> ssh -l remote-user@CEO
remote-user@ce0's password:
--- JUNOS 8.4R1.3 built 2007-08-06 06:58:15 UTC
remote-user@CEO>
```

These examples show the conventions that will appear in this book when command-line procedures are shown. All prompts, output, and code listings appear `like this`. Whenever a user types a command to produce some output, the command typed will appear **like this**. We'll see CLI examples from Windows hosts as well.

Illustrated Network Router Roles

The intermediate systems or network nodes used on the Illustrated Network are routers. Not all of the routers play the same role in the network, and some have switching capabilities. The router's role depends on its position in the network. Generally, smaller routers populate the edge of the network near the LANs and hosts, while larger routers populate the ISP's network core. The routers on our network have one of three network-centric designations; we have LAN switches also, but these are not routers.

- **Customer edge (CE):** These two routers belong to us, in our role as the customer who owns and operates the hosts and LANs. These CE routers are smaller than the other routers in terms of size, number of ports, and capabilities. Technically, on this network, they perform a gateway role.
- **Provider edge (PE):** These two routers gather the traffic from customers (typically there are many CE routers, of course). They are not usually accessible by customers.
- **Provider (P):** These six routers are arranged in what is often called a "quad." The two service providers on the Illustrated Network each manage two providers' routers in their network core. Quads make sure traffic flows smoothly even if any one router or one link fails on the provider's core networks.
- **Ethernet LAN switches:** The network also contains two Ethernet LAN switches. We'll spend a lot of time exploring switches later. For now, consider that switches operate on Layer 2 frames and routers operate on Layer 3 packets.

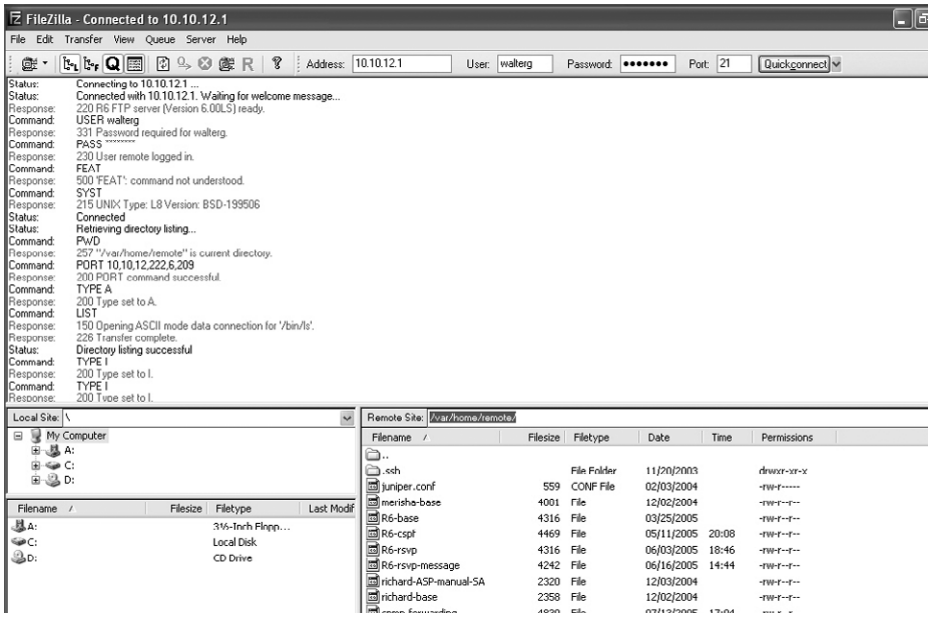
Now, what is this second example telling us? First of all, it tells us that routers, just like ordinary hosts, will allow a remote user to log in if they have the correct user ID and password. It would appear that routers aren't all that much different from hosts. However, this can be a little misleading. Hosts generally have different roles in a network than routers. For now, we'll just note that for security reasons, you don't want it to be easy for people to remotely access routers, because intruders can cause a lot of damage after compromising just a single router. In practice, a lot more security than just passwords is employed to restrict router access.

Secure remote access to a router is usually necessary, so not running the *process* or *entity* that allows remote access isn't an option. An organization with a large network could have routers in hundreds of locations scattered all over the country (or even the world). These devices need *management*, which includes tasks such as changing the configuration of the routers. Router configuration often includes details about the protocols' operation and other capabilities of the router, which can change as the network evolves. Software upgrades need to be distributed as well. Troubleshooting procedures often require direct entry of commands to be executed on the router. In short, remote access and file transfer can be very helpful for router and network management purposes.

File Transfer to a Router

Let's look at the transfer of a new router configuration file, for convenience called `routerconfig.txt`, from a client host (`wincli2`) to router `CE0`. This time we'll use a GUI for the file transfer protocol (FTP) application, which will be shown as a figure, as in Figure 1.2. First, we have to remotely access the router.

The main window section in the figure shows remote access and the file listing of the default directory on the router, which is `/var/home/remote` (the router uses the Unix file system). The listing in the lower right section is the contents of the default


FileZilla - Connected to 10.10.12.1
File Edit Transfer View Queue Server Help
Address: 10.10.12.1
User: walterg
Password: •••••••
Port: 21
Quickconnect

```
Status:     Connecting to 10.10.12.1 ...
Status:     Connected with 10.10.12.1. Waiting for welcome message...
Response:   220 R6 FTP server (Version 6.00LS) ready.
Command:    USER walterg
Response:   331 Password required for walterg.
Command:    PASS ********
Response:   230 User remote logged in.
Command:    FEAT
Response:   500 'FEAT': command not understood.
Command:    SYST
Response:   215 UNIX Type: L8 Version: BSD-199506
Status:     Connected
Status:     Retrieving directory listing...
Command:    PWD
Response:   257 "/var/home/remote" is current directory.
Command:    PORT 10,10,12,222,6,209
Response:   200 PORT command successful.
Command:    TYPE A
Response:   200 Type set to A.
Command:    LIST
Response:   150 Opening ASCII mode data connection for '/bin/ls'.
Response:   226 Transfer complete.
Status:     Directory listing successful
Command:    TYPE I
Response:   200 Type set to I.
Command:    TYPE I
Response:   200 Type set to I.
```

Local Site: \

```
My Computer
  A:
  C:
  D:
```

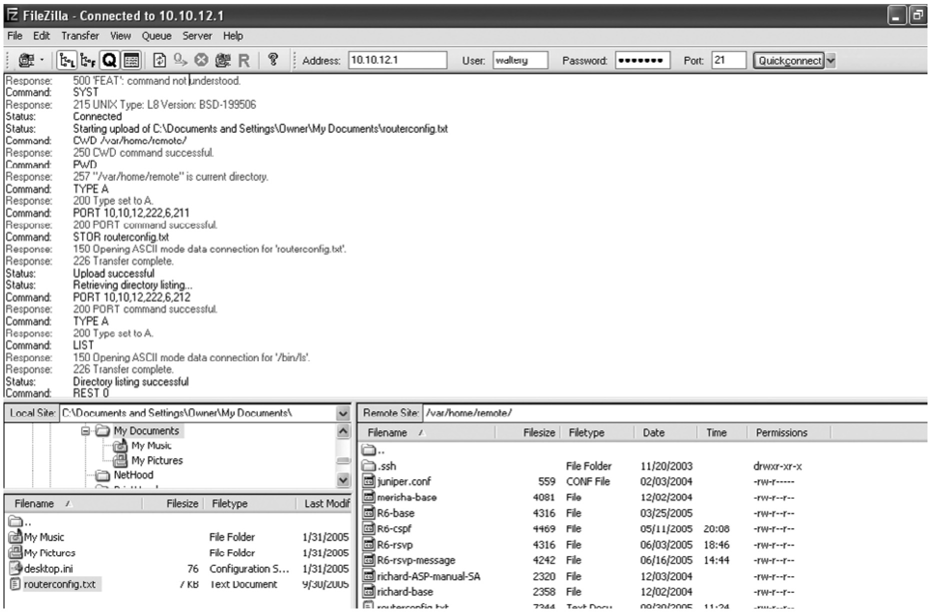
Filename	Filesize	Filetype	Last Modif
A:		3½-Inch Flopp...	
C:		Local Disk	
D:		CD Drive	

Remote Site: /var/home/remote/

Filename	Filesize	Filetype	Date	Time	Permissions
..					
.ssh		File Folder	11/20/2003		drwxr-xr-x
juniper.conf	559	CONF File	02/03/2004		-rw-r-----
merisha-base	4001	File	12/02/2004		-rw-r--r--
R6-base	4316	File	03/25/2005		-rw-r--r--
R6-cspf	4469	File	05/11/2005	20:08	-rw-r--r--
R6-rsvp	4316	File	06/03/2005	18:46	-rw-r--r--
R6-rsvp-message	4242	File	06/16/2005	14:44	-rw-r--r--
richard-ASP-manual-SA	2320	File	12/03/2004		-rw-r--r--
richard-base	2358	File	12/02/2004		-rw-r--r--

FIGURE 1.2

Remote access for FTP using a GUI. Note how the different panes give different types of information, yet bring it all together.


FileZilla - Connected to 10.10.12.1
File Edit Transfer View Queue Server Help
Address: 10.10.12.1 User: walterg Password: ••••••• Port: 21 Quickconnect

```
Response:  500 'FEAT': command not understood.
Command:   SYST
Response:  215 UNIX Type: L8 Version: BSD-199506
Status:    Connected
Status:    Starting upload of C:\Documents and Settings\Owner\My Documents\routerconfig.txt
Command:   CWD /var/home/remote/
Response:  250 CWD command successful.
Command:   PWD
Response:  257 "/var/home/remote" is current directory.
Command:   TYPE A
Response:  200 Type set to A.
Command:   PORT 10,10,12,222,6,211
Response:  200 PORT command successful.
Command:   STOR routerconfig.txt
Response:  150 Opening ASCII mode data connection for 'routerconfig.txt'.
Response:  226 Transfer complete.
Status:    Upload successful
Status:    Retrieving directory listing...
Command:   PORT 10,10,12,222,6,212
Response:  200 PORT command successful.
Command:   TYPE A
Response:  200 Type set to A.
Command:   LIST
Response:  150 Opening ASCII mode data connection for '/bin/ls'.
Response:  226 Transfer complete.
Status:    Directory listing successful
Command:   REST 0
```

```
Local Site: C:\Documents and Settings\Owner\My Documents\
    My Documents
        My Music
        My Pictures
    NetHood

Filename          Filesize  Filetype          Last Modif
..
My Music                    File Folder       1/31/2005
My Pictures                 File Folder       1/31/2005
desktop.ini             76  Configuration S... 1/31/2005
routerconfig.txt      7 KB  Text Document     9/30/2005
```

```
Remote Site: /var/home/remote/

Filename               Filesize  Filetype     Date        Time   Permissions
..
.ssh                             File Folder  11/20/2003         drwxr-xr-x
juniper.conf                559  CONF File    02/03/2004         -rw-r-----
merisha-base               4081  File         12/02/2004         -rw-r--r--
R6-base                    4316  File         03/25/2005         -rw-r--r--
R6-cspf                    4469  File         05/11/2005  20:08  -rw-r--r--
R6-rsvp                    4316  File         06/03/2005  18:46  -rw-r--r--
R6-rsvp-message            4242  File         06/16/2005  14:44  -rw-r--r--
richard-ASP-manual-SA      2320  File         12/03/2004         -rw-r--r--
richard-base               2358  File         12/02/2004         -rw-r--r--
```

FIGURE 1.3

File transfer with a GUI. There are commands (user mouse clicks that trigger messages), responses (the server's replies), and status lines (reports on the state of the interaction).

directory, not part of the command/response dialog between host and router. The lower left section shows the file system on the host, which is a Windows system. Note that the file transfer is not encrypted or secured in any way.

Most "traditional" Unix-derived TCP/IP applications have both CLI and GUI interfaces available, and which one is used is usually a matter of choice. Older Unix systems, the kind most often used on the early Internet, didn't typically have GUI interfaces, and a lot of users prefer the CLI versions, especially for book illustrations. GUI applications work just as well, and don't require users to know the individual commands well. When using the GUI version of FTP, all the user has to do is "drag and drop" the local `routerconfig.txt` file from the lower left pane to the lower right pane of the window to trigger the commands (which the application produces "automatically") for the transfer to occur. This is shown in Figure 1.3.

With the GUI, the user does not have to issue any FTP commands directly.

CLI and GUI

We'll use both the CLI and GUI forms of TCP/IP applications in this book. In a nod to tradition, we'll use the CLI on the Unix systems and the GUI versions when Windows systems are used in the examples. (CLI commands often capture details that are not easily seen in GUI-based applications.) Keep in mind that you can use GUI applications on Unix and the CLI on Windows (you have to run `cmd` first to access the Windows CLI). This listing shows the router configuration file transfer of `newrouterconfig.txt` from the Windows XP system to router `CE6`, but with the Windows CLI and using the IP address of the router.

```
C:\Documents and Settings\Owner> ftp 10.10.12.1
Connected to 10.10.12.1.
220 R6 FTP server (version 6.00LS) ready.
User (10.10.12.1:none)):walterg
331 Password required for walterg.
Password: ********
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for '/bin/ls'.
total 128
drwxr-xr-x 2 remote staff 512 Nov 20 2004 .ssh
-rw-r--r-- 1 remote staff 4316 Mar 25 2006 R6-base
-rw-r--r-- 1 remote staff 4469 May 11 20:08 R6-cspf
-rw-r--r-- 1 remote staff 4316 Jun 3 18:46 R6-rsvp
-rw-r--r-- 1 remote staff 4242 Jun 16 14:44 R6-rsvp-message
-rw-r----- 1 remote staff 559 Feb 3 2005 juniper.conf
-rw-r--r-- 1 remote staff 4081 Dec 2 2005 merisha-base
-rw-r--r-- 1 remote staff 2320 Dec 3 2005 richard-ASP-manual-SA
-rw-r--r-- 1 remote staff 2358 Dec 2 2005 richard-base
-rw-r--r-- 1 remote staff 7344 Sep 30 11:28 routerconfig.txt
-rw-r--r-- 1 remote staff 4830 Jul 13 17:04 snmp-forwarding
-rw-r--r-- 1 remote staff 3190 Jan 7 2006 tp6
-rw-r--r-- 1 remote staff 4315 May 5 12:49 wjg-ORA-base-TP6
-rw-r--r-- 1 remote staff 4500 May 6 09:47 wjg-tp6-with-ipv6
-rw-r--r-- 1 remote staff 4956 May 8 13:42 wjg-with-ipv6
226 transfer complete
ftp: 923 bytes received in 0.00Seconds 923000.00Kbytes/sec.
ftp> bin
200 Type set to I
ftp> put newrouterconfig.text
200 PORT command successful.
150 Opening ASCII mode data connection for "newrouterconfig.txt".
226 Transfer complete.
ftp: 7723 bytes received in 0.00Seconds 7344000.00Kbytes/sec.
ftp>_
```

In some cases, we'll list CLI examples line by line, as here, and in other cases we will show them in a figure.

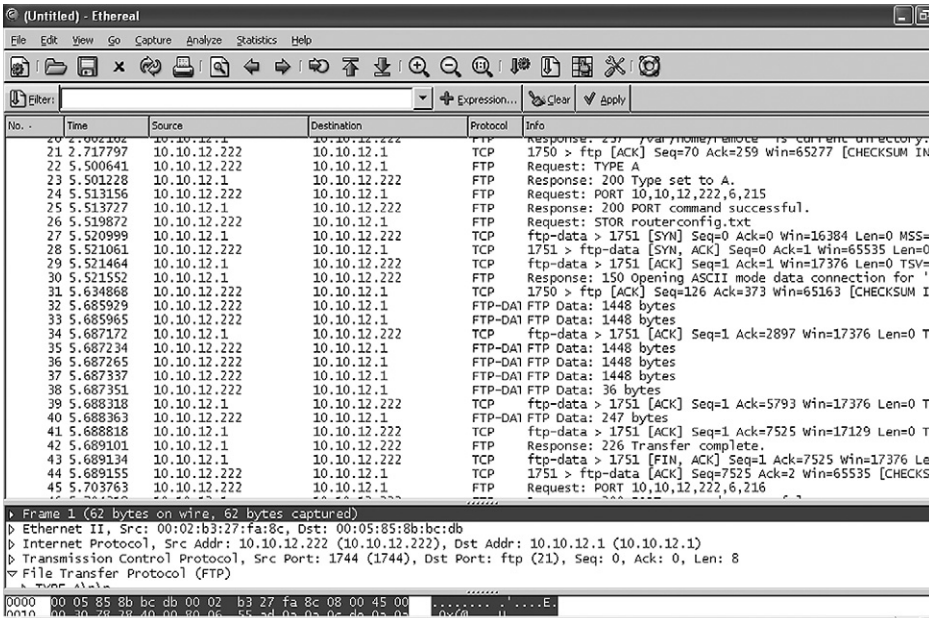
Ethereal and Packet Capture

Of course, showing a GUI or command line FTP session doesn't reveal much about how the network functions. We need to look at the bits that are flowing through the network. Also, we need to look at applications, such as the file transfer protocol, from the network perspective.

To do so, we'll use a packet capture utility. This book will use the Ethereal packet capture program in fact and by name throughout, although shortly after the project began, Ethereal became Wireshark. The software is the same, but all development will now be done through the Wireshark organization. Wireshark (Ethereal) is available free of charge at *www.wireshark.org*. It is notable that Wireshark, unlike a lot of similar applications, is available for Windows as well as most Unix/Linux variations.

Ethereal is a network protocol analyzer program that keeps a copy of every packet of information that emerges from or enters the system on a particular interface. Ethereal also parses the packet and shows not only the bit patterns, but what those bit groupings mean. Ethereal has a summary screen, a pane for more detailed information, and a pane that shows the raw bits that Ethereal captured. The nicest feature of Ethereal is that the packet capture stream can be saved in a standard *libpcap* format file (usually with a `.cap` or `.pcap` extension), which is common among most protocol analyzers. These files can be read and parsed and replayed by `tcpdump` and other applications or Ethereal on other systems.

Figure 1.4 shows the same router configuration file transfer as in Figure 1.2 and 1.3, and at the same time. However, this time the capture is not at the user level, but at the network level.



```
No. -  Time      Source          Destination     Protocol Info
    21 2.717797  10.10.12.222    10.10.12.1      TCP      1750 > ftp [ACK] Seq=70 Ack=259 Win=65277 [CHECKSUM IN
    22 5.500641  10.10.12.222    10.10.12.1      FTP      Request: TYPE A
    23 5.501228  10.10.12.1      10.10.12.222    FTP      Response: 200 Type set to A.
    24 5.513156  10.10.12.222    10.10.12.1      FTP      Request: PORT 10,10,12,222,6,215
    25 5.513727  10.10.12.1      10.10.12.222    FTP      Response: 200 PORT command successful.
    26 5.519872  10.10.12.222    10.10.12.1      FTP      Request: STOR routerconfig.txt
    27 5.520999  10.10.12.1      10.10.12.222    TCP      ftp-data > 1751 [SYN] Seq=0 Ack=0 Win=16384 Len=0 MSS=
    28 5.521061  10.10.12.222    10.10.12.1      TCP      1751 > ftp-data [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
    29 5.521464  10.10.12.1      10.10.12.222    TCP      ftp-data > 1751 [ACK] Seq=1 Ack=1 Win=17376 Len=0 TSV=
    30 5.521552  10.10.12.1      10.10.12.222    FTP      Response: 150 Opening ASCII mode data connection for '
    31 5.634868  10.10.12.222    10.10.12.1      TCP      1750 > ftp [ACK] Seq=126 Ack=373 Win=65163 [CHECKSUM I
    32 5.685929  10.10.12.222    10.10.12.1      FTP-DA1 FTP Data: 1448 bytes
    33 5.685965  10.10.12.222    10.10.12.1      FTP-DA1 FTP Data: 1448 bytes
    34 5.687172  10.10.12.1      10.10.12.222    TCP      ftp-data > 1751 [ACK] Seq=1 Ack=2897 Win=17376 Len=0 T
    35 5.687234  10.10.12.222    10.10.12.1      FTP-DA1 FTP Data: 1448 bytes
    36 5.687265  10.10.12.222    10.10.12.1      FTP-DA1 FTP Data: 1448 bytes
    37 5.687337  10.10.12.222    10.10.12.1      FTP-DA1 FTP Data: 1448 bytes
    38 5.687351  10.10.12.222    10.10.12.1      FTP-DA1 FTP Data: 36 bytes
    39 5.688318  10.10.12.1      10.10.12.222    TCP      ftp-data > 1751 [ACK] Seq=1 Ack=5793 Win=17376 Len=0 T
    40 5.688363  10.10.12.222    10.10.12.1      FTP-DA1 FTP Data: 247 bytes
    41 5.688818  10.10.12.1      10.10.12.222    TCP      ftp-data > 1751 [ACK] Seq=1 Ack=7525 Win=17129 Len=0 T
    42 5.689101  10.10.12.1      10.10.12.222    FTP      Response: 226 Transfer complete.
    43 5.689134  10.10.12.1      10.10.12.222    TCP      ftp-data > 1751 [FIN, ACK] Seq=1 Ack=7525 Win=17376 Le
    44 5.689155  10.10.12.222    10.10.12.1      TCP      1751 > ftp-data [ACK] Seq=7525 Ack=2 Win=65535 [CHECKS
    45 5.703763  10.10.12.222    10.10.12.1      FTP      Request: PORT 10,10,12,222,6,216

▸ Frame 1 (62 bytes on wire, 62 bytes captured)
▷ Ethernet II, Src: 00:02:b3:27:fa:8c, Dst: 00:05:85:8b:bc:db
▷ Internet Protocol, Src Addr: 10.10.12.222 (10.10.12.222), Dst Addr: 10.10.12.1 (10.10.12.1)
▷ Transmission Control Protocol, Src Port: 1744 (1744), Dst Port: ftp (21), Seq: 0, Ack: 0, Len: 8
▽ File Transfer Protocol (FTP)

0000  00 05 85 8b bc db 00 02  b3 27 fa 8c 08 00 45 00   ........ .'....E.
```

FIGURE 1.4

Ethereal FTP capture of the file transfer shown earlier from the user perspective.

Each packet captured is numbered sequentially and given a time stamp, and its source and destination address is listed. The protocol is in the next column, followed by the interpretation of the packet's meaning and function. The packet to request the router to `STOR routerconfig.txt` is packet number 26 in the sequence.

Already we've learned something important: that with TCP/IP, the number of packets exchanged to accomplish even something basic and simple can be surprisingly large. For this reason, in some cases, we'll only show a section of the panes of the full Ethereal screen, only to cut down on screen clutter. The captured files are always there to consult later.

With these tools—CLI listings, GUI figures, and Ethereal captures—we are prepared to explore all aspects of modern network operation using TCP/IP.

First Explorations in Networking

We've already seen that an authorized user can access a router from a host. We've seen that routers can run the `ssh` and `ftp` server applications `sshd` and `ftpd`, and the suspicion is that they might be able to run even more (they can just as easily be `ssh` and `ftp` clients). However, the router application suite is fairly restrictive. You usually don't, for example, send email to a router, or log in to a router and then browse Web sites. There is a fundamental difference in the roles that hosts and routers play in a network. A router doesn't have all of the application software you would expect to find on a client or server, and a router uses them mainly for management purposes. However, it does have all the layers of the protocol suite.

TCP/IP networks are a mix of hosts and routers. Hosts often talk to other devices on the network, or expose their applications to the network, but their basic function is to run programs. However, network systems like routers exist to keep the network running, which is their primary task. Router-based applications support this task, although in theory, routers only require a subset of the TCP/IP protocol suite layers to perform their operational role. You also have to manage routers, and that requires some additional software in practice. However, don't expect to find chat or other common client applications on a router.

What is it about protocols and layers that is so important? That's what the rest of this chapter is about. Let's start with what protocols are and where they come from.

PROTOCOLS

Computers are systems or devices capable of running a number of processes. These processes are sometimes referred to as *entities*, but we'll use the term processes. Computer networks enable communication between processes on two different devices that are capable of sending and receiving information in the form of bits (0s and 1s). What pattern should the exchange of bits follow? Processes that exchange bit streams must agree on a *protocol*. A protocol is a set of rules that determines all aspects of data communication.

A protocol is a standard or convention that enables and controls the connection, communication, and transfer of information between two communications endpoints, or hosts. A protocol defines the rules governing the syntax (what can be communicated), semantics (how it can be communicated), and synchronization (when and at what speed it can be communicated) of the communications procedure. Protocols can be implemented on hardware, software, or a combination of both.

Protocols are not the same as standards: some standards have never been implemented as workable protocols, while some of the most useful protocols are only loosely defined (this sometimes makes interconnection an adventure). The protocols discussed in this book vary greatly in degree of sophistication and purpose. However, most of the protocols specify one or more of the following:

Physical connection—The host typically uses different hardware depending on whether the connection is wired or wireless, and some other parameters might require manual configuration. However, protocols are used to supply details about the network connection (speed is part of this determination). The host can usually detect the presence (or absence) of the other endpoint devices as well.

Handshaking—A protocol can define the rules for the initial exchange of information across the network.

Negotiation of parameters—A protocol can define a series of actions to establish the rules and limits used for communicating across the network.

Message delimiters—A protocol can define what will constitute the start and end of a message on the network.

Message format—A protocol can define how the content of a message is structured, usually at the "field" level.

Error detection—A protocol can define how the receiver can detect corrupt messages, unexpected loss of connectivity, and what to do next. A protocol can simply fail or try to correct the error.

Error correction—A protocol can define what to do about these error situations. Note that error recovery usually consists of both error-detection and error-correction protocols.

Termination of communications—A protocol can define the rules for gracefully stopping communicating endpoints.

Protocols at various layers provided the abstraction necessary for Internet success. Application developers did not have to concern themselves overly with the physical properties of the network. The expanded use of communications protocols has been a major contributor to the Internet's success, acceptance, flexibility, and power.

Standards and Organizations

Anyone can define a protocol. Simply devise a set of rules for any or all of the phases of communication and convince others to make hardware or software that implements the new method. Of course, an implementer could try to be the only source of a given protocol, a purely *proprietary* situation, and this was once a popular way to develop protocols. After all, who knew better how to network IBM computers than IBM? Today, most closed protocols have given way to *open* protocols based on published *standards*, especially since the Internet strives for connectivity between all types of computers and related devices and is not limited to equipment from a certain vendor. Anyone who implements an open protocol correctly from public documents should in most cases be able to interoperate with other versions of the same protocol.

Standards promote and maintain an open and competitive market for network hardware and software. The overwhelming need for *interoperability* today, both nationally and internationally, has increased the set of choices in terms of vendor and capability for each aspect of data communications. However, proprietary protocols intended for a limited architecture or physical network are still around, of course. Proprietary protocols might have some very good application-specific protocols, but could probably not support things like the Web as we know it. Making something a standard does not guarantee market acceptance, but it is very difficult for a protocol to succeed without a standard for everyone to follow. Standards provide essential guidelines to manufacturers, vendors, service providers, consultants, government agencies, and users to make sure the interconnectivity needed today is there.

Data communication standards fall into two major categories: *de jure* ("by rule or regulation") and *de facto* ("by fact or convention").

De jure—These standards have been approved by an officially recognized body whose job is to standardize protocols and other aspects of networking. *De jure* standards often have the force of law, even if they are called *recommendations* (for these basic standards, it is recommended that nations use their own enforcement methods, such as fines, to make sure they are followed).

De facto—Standards that have *not* been formally approved but are widely followed fall into this category. If someone wants to do something different, such as a manufacturer of network equipment, this method can be used to quickly establish a new product or technology. These types of standards can always be proposed for *de jure* approval.

When it comes to the Internet protocols, things are a bit more complicated. There are very few official standards, and there are no real penalties involved for not following them (other than the application not working as promised). On the Internet, a "*de facto* standard" forms a *reference implementation* in this case. *De facto* standards are also often subportions or implementation details for formal standards, usually when the formal standard falls short of providing all the information needed to create a working program. Internet standard proposals in many cases require running code at some stages of the process: at least the *de facto* code will cover the areas that the standard missed.

The standards for the TCP/IP protocol suite now come from the Internet Engineering Task Force (IETF), working in conjunction with other Internet organizations. The IETF is neither strictly a de facto nor de jure standards organization: There is no force of law behind Internet standards; they just don't work the way they should if not done correctly. We'll look at the IETF in detail shortly. The Internet uses more than protocol standards developed by the IETF. The following organizations are the main ones that are the sources of these other standards.

Institute of Electrical and Electronics Engineers

This international organization is the largest society of professional engineers in the world. One of its jobs is to oversee the development and adaptation of international standards, often in the local area network (LAN) arena. Examples of IEEE standards are all aspects of wireless LANs (IEEE 802.11).

American National Standards Institute

Although ANSI is actually a private nonprofit organization, and has no affiliation with the federal government, its goals include serving as the national institution for coordinating voluntary standardization in the United States as a way of advancing the U.S. economy and protecting the public interest. ANSI's members are consumer groups, government and regulatory bodies, industry associations, and professional societies. Other countries have similar organizations that closely track ANSI's actions. The indispensable American Standard Code for Information Interchange (ACSII) that determines what bits mean is an example of an ANSI standard.

Electronic Industries Association

This is a nonprofit organization aligned with ANSI to promote electronic manufacturing concerns. The EIA has contributed to networking by defining physical connection interfaces and specifying electrical signaling methods. The popular Recommended Jack #45 (RJ-45) connector for twisted pair LANs is an example of an EIA standard.

ISO, or International Standards Organization

Technically, this is the International Organization for Standardization in English, one of its official languages, but is always called the *ISO*. "ISO" is not an acronym or initialism for the organization's full name in either English or French (its two official languages). Rather, the organization adopted ISO based on the Greek word *isos*, meaning *equal*. Recognizing that the organization's initials would vary according to language, its founders chose ISO as the universal short form of its name. This, in itself, reflects the aim of the organization: to equalize and standardize across cultures. This multinational body's members are drawn from the standards committees of various governments. They are a voluntary organization dedicated to agreement on worldwide standards. The ISO's major contribution in the field of networking is with the creation of a *model* of data networking, the *Open Systems Interconnection Reference Model (ISO-RM)*, which also forms the basis for a working set of protocols. The United States is represented by ANSI in the ISO.

International Telecommunications Union–Telecommunication Standards Sector

A global economy needs international standards not only for data networks, but for the global *public switched telephone network* (PSTN). The United Nations formed a committee under the International Telecommunications Union (ITU), known as the Consultative Committee for International Telegraphy and Telephony (CCITT), that was eventually reabsorbed into the parent body as the ITU-T in 1993. All communications that cross national boundaries must follow ITU-T "recommendations," which have the force of law. However, inside a nation, local standards can apply (and usually do). A network architecture called asynchronous transfer mode (ATM) is an example of an ITU-T standard.

In addition to these standards organizations, networking relies on various *forums* to promote new technologies while the standardization process proceeds at the national and international levels. Forum members essentially pledge to follow the specifications of the forum when it comes to products, services, and so forth, although there is seldom any penalty for failing to do so. The Metro Ethernet Forum (MEF) is a good example of the modern forum in action.

The role of regulatory agencies cannot be ignored in standard discussions. It makes no sense to develop a new service for wireless networking in the United States, for example, if the Federal Communications Commission (FCC) has forbidden the use of the frequencies used by the new service for that purpose. Regulated industries include radio, television, and wireless and cable systems.

Request for Comment and the Internet Engineering Task Force

What about the Internet itself? The Internet Engineering Task Force (IETF) is the organization directly responsible for the development of Internet standards. The IETF has its own system for standardizing network components. In particular, Internet standards cover many of the protocols used by devices attached to the Internet, especially those closer to the user (applications) than to the physical network.

Internet standards are formalized regulations followed and used by those who work on the Internet. They are specifications that have been tested and must be followed. There is a strict procedure that all Internet components follow to become standards. A specification starts out as an *Internet draft*, a working document that often is revised, has no official status, and has a 6-month life span. Developers often work from these drafts, and much can be learned from the practical experience of implementation of a draft. If recommended, the Internet authorities can publish the draft as a *request for comment (RFC)*. The term is historical, and does not imply that feedback is required (most of the feedback is provided in the drafting process). Each RFC is edited, assigned a number, and available to all. Not all RFCs are standards, even those that define protocols.

This book will make heavy use of RFCs to explain all aspects of TCP/IP and the Internet, so a few details are in order. RFCs have various *maturity levels* that they go through in their lifetimes, according to their *requirement levels*. The RFC life-cycle maturity levels are shown in Figure 1.5. Note that the timeline does not always apply, or is not applied in a uniform fashion.

A specification can fall into one of six maturity levels, after which it passes to historical status and is useful only for tracking a protocol's development. Following introduction as an Internet draft, the specification can be a:

Proposed standard—The specification is now well understood, stable, and sufficiently interesting to the Internet community. The specification is now usually tested and implemented by several groups, if this has not already happened at the draft level.

Draft standard—After at least two successful and independent implementations, the proposed standard is elevated to a draft standard. Without complications, and with modifications if specific problems are uncovered, draft standards normally become Internet standards.

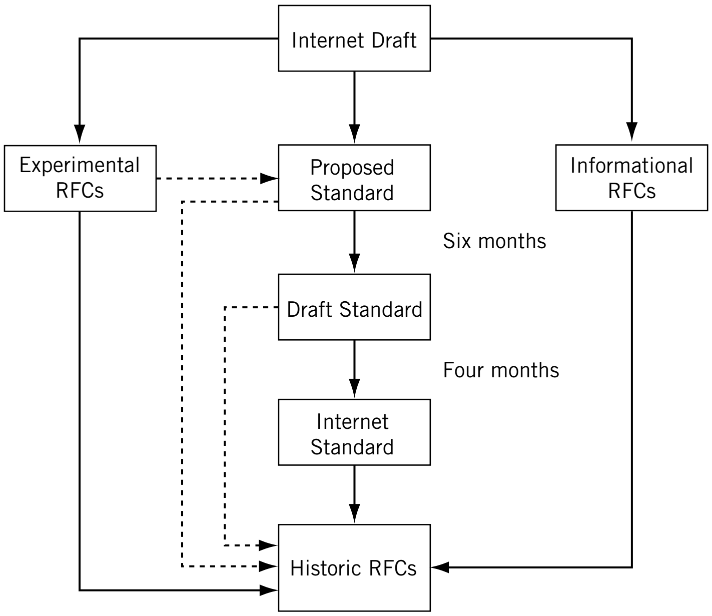


FIGURE 1.5

The RFC life cycle. Many experimental RFCs never make it to the standards track.

Internet standard—After demonstrations of successful implementation, a draft standard becomes an Internet standard.

Experimental RFCs—Not all drafts are intended for the "standards track" (and a huge number are not). Work related to an experimental situation that does affect Internet operation comprise experimental RFCs. These RFCs should *not* be implemented as part of any functional Internet service.

Informational RFCs—Some RFCs contain general, historical, or tutorial information rather than instructions.

RFCs are further classified into one of five requirement levels, as shown in Figure 1.6.

Required—These RFCs must be implemented by all Internet systems to ensure minimum conformance. For example, IPv4 and ICMP, both discussed in detail in this book, are required protocols. However, there are *very* few required RFCs.

Recommended—These RFCs are not required for minimum conformance, but are very useful. For example, FTP is a recommended protocol.

Elective—RFCs in this category are not required and not recommended. However, systems can use them for their benefit if they like, so they form a kind of "option set" for Internet protocols.

Limited Use—These RFCs are only used in certain situations. Most experimental RFCs are in this category.

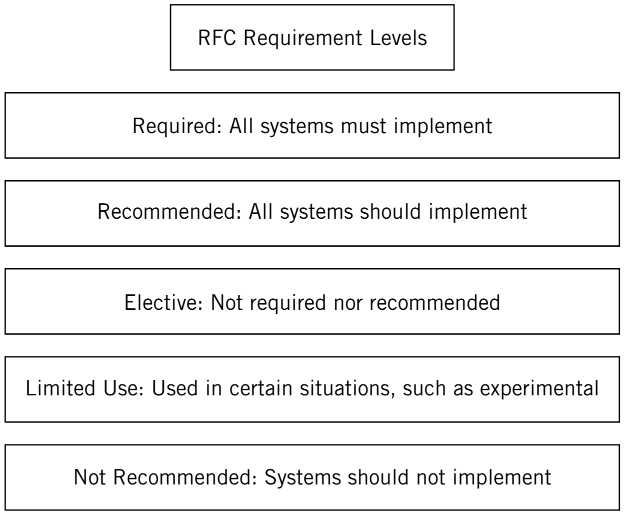

RFC Requirement Levels
Required: All systems must implement
Recommended: All systems should implement
Elective: Not required nor recommended
Limited Use: Used in certain situations, such as experimental
Not Recommended: Systems should not implement

FIGURE 1.6

RFC requirement levels. There are very few RFCs that are required to implement an Internet protocol suite.

Not Recommended—These RFCs are inappropriate for general use. Most historic (obsolete) RFCs are in this category.

RFCs can be found at *www.rfc-editor.org/rfc.html*. Current Internet drafts can be found at *www.ietf.org/ID.html*. *Expired* Internet drafts can be found at *www.watersprings.org/pub/id/index-all.html*.

INTERNET ADMINISTRATION

As the Internet has evolved from an environment with a large student user population to a more commercialized network with a broad user base, the groups that have guided and coordinated Internet issues have evolved. Figure 1.7 shows the general structure of the Internet administration entities.

Internet Society (ISOC)—This is an international nonprofit organization formed in 1992 to support the Internet standards process. ISOC maintains and supports the other administrative bodies described in this section. ISOC also supports research and scholarly activities relating to the Internet.

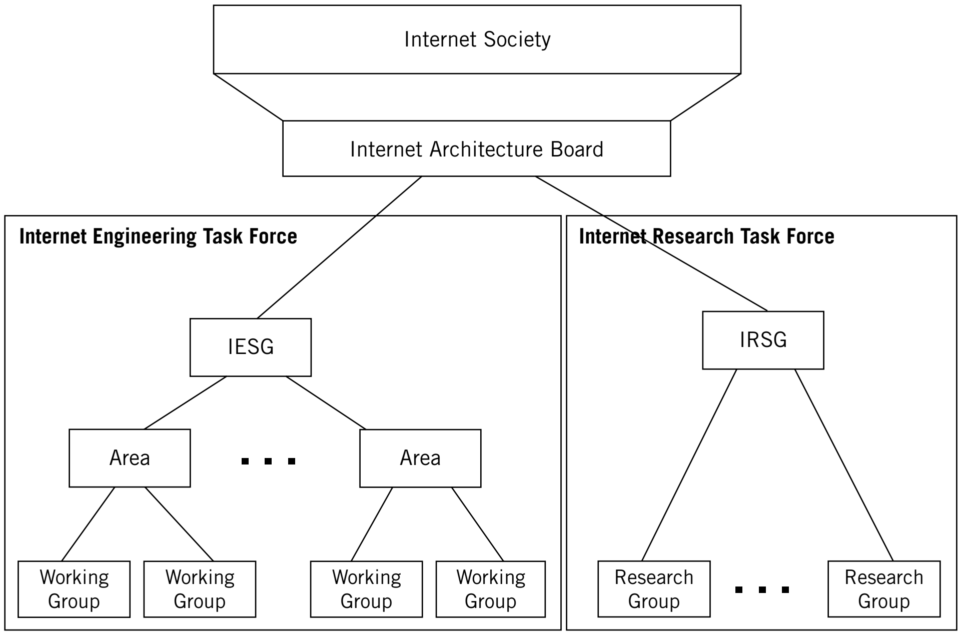


FIGURE 1.7

Internet administration groups, showing the interactions between the major components.

Internet Architecture Board (IAB)—This group is the technical advisor to ISOC. The IAB oversees the continued development of the Internet protocol suite and plays a technical advisory role to members of the Internet community involved in research. The IAB does this primarily through the two organizations under it. In addition, the RFC editor derives authority from the IAB, and the IAB represents the Internet to other standards organizations and forums.

Internet Engineering Task Force (IETF)—This a forum of *working groups* managed by the Internet Engineering Steering Group (IESG). The IETF identifies operational problem areas and proposes solutions. They also develop and review the specifications intended to become Internet standards. The working groups are organized into areas devoted to a particular topic. Nine areas have been defined, although this can change: applications, Internet protocols, routing, operations, user services, network management, transport, IPv6, and security. The IETF has taken on some of the roles that were invested in ISOC.

Internet Research Task Force (IRTF)—This is another forum of working groups, organized directly under the Internet Research Steering Group (IESG) for management purposes. The IRTF is concerned with long-term research topics related to Internet protocols, applications, architecture, and technology.

Two other groups are important for Internet administration, although they do not appear in Figure 1.7.

Internet Corporation for Assigned Names and Numbers (ICANN)—This is a private nonprofit corporation that is responsible for the management of all Internet *domain names* (more on these later) and Internet addresses. Before 1998, this role was played by the Internet Assigned Numbers Authority (IANA), which was supported by the U.S. government.

Internet Network Information Center (InterNIC)—The job of the InterNIC, run by the U.S. Department of Commerce, is to collect and distribute information about IP names and addresses. They are at *http://www.internic.net*.

LAYERS

When it comes to communications, all of these standard organizations have one primary function: the creation of standards that can be combined with others to create a working network. One concern is that these organizations be able to recommend solutions that are both flexible and complete, even though no single standards entity has complete control over the entire process from top to bottom. The way this is done is to divide the communications process up into a number of functional layers.

Data communication networks rely on *layered* protocols. In brief, processes running on a system and the communication ports that send and receive network bits are logically connected by a series of layers, each performing one major function of the networking task.

The key concept is that each layer in the protocol stack has a distinct purpose and function. There is a big difference between the application layer protocols we've seen, such as FTP and SSH, and a lower-level protocol such as Ethernet on a LAN. Each protocol layer handles part of the overall task.

For example, Ethernet cards format the bits sent out on a LAN at one layer, and FTP client software communicates with the FTP server at a higher layer. However, the Ethernet card does not tell the FTP application which bits to send out the interface. FTP addresses the higher-end part of the puzzle: sending commands and data to the FTP server. Other layers take care of things like formatting, and can vary in capability or form to address differences at every level. You don't use different Web browsers depending on the type of links used on a network. The whole point is that not all networks are Ethernet (for example), so a layered protocol allows a "mix and match" of whatever protocols are needed for the network at each layer.

Simple Networking

Most programming languages include statements that allow the programmer to send bits out of a physical connector. For example, suppose a programming language allowed you to program a statement like `write(port 20$, "test 1")`. Sure enough, when compiled, linked, and run, the program would spit the bits representing the string "test 1" out the communications port of the computer. A similar statement like `read(port 20$, STUFF)` would, when compiled, linked, and run, wait until something appeared in the buffer of the serial port and store the bits in the variable called STUFF.

A simple network using this technique is shown in Figure 1.8. (There is still some software in use that does networking this way.)

However, there are some things to consider. Is there anything attached to the port at all? Or are the bits just falling into the "bit bucket"? If there was a link attached, what if someone disconnected it while the bits are in flight? What about other types of errors? How would we know that the bits arrived safely?

Even assuming that the bits got there, and some listening process received them, does the content make sense? Some computers store bits differently than others, and "test 1" could be garbled on the other system. How many bits are sent to represent the

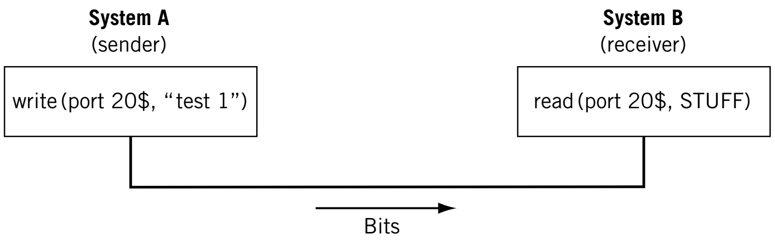


FIGURE 1.8

An extremely simple network with a distinctly non-layered approach to networking.

number 1? How do we know that a "short integer" used by the sender is the same as the "short integer" used by another? (In fairness, TCP/IP does little to address this issue directly.)

We see that the networking task is not as simple as it seems. Now, each and every networked application program could conceivably include every line of code that is needed to solve all of these issues (and there are even others), but that introduces another factor into the networking equation. Most hosts attached to a network have only one communications port active at any one time (the "network interface"). If an "all-in-one" network application is using it, perhaps to download a music file, how can another application use the same port for email? It can't.

Besides the need to multiplex in various ways, another factor influencing layers is that modern operating systems do not allow direct access to hardware. The need to go through the operating system and multiplex the network interface leads to a centralization of the networking tasks in the end system.

Protocol layers make all of these issues easier to deal with, but they cannot be added haphazardly. (You can still create a huge and ugly "layer" that implements everything from hardware to transport to data representation, but it would work.) As important as the layers are, the tasks and responsibilities assigned to those layers are even *more* important.

Protocol Layers

Each layer has a separate function in the overall task of moving bits between processes. These processes could be applications on separate systems, but on modern systems a lot of process-to-process communication is not host-to-host. For example, a lot of printer management software runs as a Web browser using a special loopback TCP/IP address to interface with the process that gathered status information from the printer.

As long as the boundary functions between adjacent layers are respected, layers can be changed or even completely rewritten without having to change the whole application. Layers can be combined for efficiency, "mixed-and-matched" from different vendors, or customized for different circumstances, all without having to rework the entire stack from top to bottom.

Nearly every layer has some type of multiplexing field to allow the receiver to determine the type of *payload*, or content of the data unit at a particular layer. A key point in networking is that the payload and control information at one layer is just a "transparent" (meaningless) payload to the layer below. Transparent bits, as the name implies, are passed unchanged to the next layer.

How can protocol layers work together? Introducing a bunch of new interfaces and protocols seems to have made the networking task harder, not easier. There is a simple method called *encapsulation* that makes the entire architecture workable. What is encapsulation? Think of the layers of the protocol suite in terms of writing a letter and the systems that are involved in letter delivery. The letter goes inside an envelope which is gathered with others inside a mailbag which is transported with others inside a truck or plane. It sounds like a very complicated way to deliver one message, but this system makes the overall task of delivering many messages easier, not harder. For example, there now can be facilities that only deal with mailbags and do not worry about an individual letter's language or the transportation details.

THE TCP/IP PROTOCOL SUITE

The protocol stack used on the Internet is the Internet Protocol Suite. It is usually called TCP/IP after two of its most prominent protocols, but there are other protocols as well. The *TCP/IP model* is based on a five-layer model for networking. From bottom (the link) to top (the user application), these are the physical, data link, network, transport, and application layers. Not all layers are completely defined by the model, so these layers are "filled in" by external standards and protocols. The layers have names but no numbers, and although sometimes people speak of "Layer 2" or "Layer 3," these are not TCP/IP terms. Terms like these are actually from the OSI Reference Model.

The TCP/IP stack is *open*, which means that there are no "secrets" as to how it works. (There are "open systems" too, but with TCP/IP, the systems do not have to be "open" and often are not.) Two compatible end-system applications can communicate regardless of their underlying architectures, although the connections between layers are not defined.

The OSI Reference Model

The TCP/IP or Internet model is not the only standard way to build a protocol suite or stack. The Open Standard Interconnection (OSI) reference model is a seven-layer model that loosely maps into the five layers of TCP/IP. Until the Web became widely popular in the 1990s, the OSI reference model, with distinctive names and numbers for its layers, was proposed as the standard model for all communication networks. Today, the OSI reference model (OSI-RM) is often used as a learning tool to introduce the functions of TCP/IP.

The TCP/IP stack is comprised of modules. Each module provides a specific function, but the modules are fairly independent. The TCP/IP layers contain relatively independent protocols that can be used depending on the needs of the system to provide whatever function is desired. In TCP/IP, each higher layer protocol is supported by lower layer protocols. The whole collection of protocols forms a type of hourglass shape, with IP in the middle, and more and more protocols up or down from there.

Suite, Stack, and Model

The term "protocol stack" is often used synonymously with "protocol suite" as an implementation of a reference model. However, the term "protocol suite" properly refers to a collection of all the protocols that can make up a layer in the reference model. The Internet protocol suite is an example of the Internet or TCP/IP reference model protocols, and a TCP/IP protocol stack implements one or more of these protocols at each layer.

The TCP/IP Layers

The TCP/IP protocol stack models a series of protocol layers for networks and systems that allows communications between any types of devices. The model consists of five separate but related layers, as shown in Figure 1.9. The Internet protocol suite is based on these five layers. TCP/IP says most about the network and transport layers, and a lot about the application layer. TCP/IP also defines how to interface the network layer with the data link and physical layers, but is not directly concerned with these two layers themselves.

The Internet protocol suite assumes that a layer is there and available, so TCP/IP does not define the layers themselves. The stack consist of protocols, not implementations, so describing a layer or protocols says almost nothing about how these things should actually be built.

Not all systems on a network need to implement all five layers of TCP/IP. Devices using the TCP/IP protocol stack fall into two general categories: a *host* or *end system* (ES) and an *intermediate node* (often a router) or an *intermediate system* (IS). The

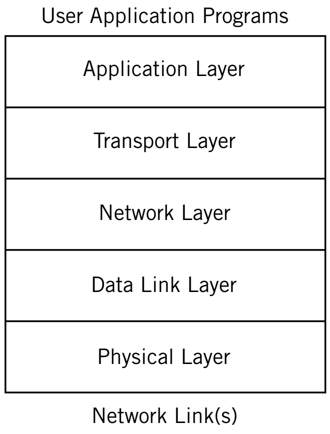


FIGURE 1.9

The five layers of TCP/IP. Older models often show only four layers, combining the physical and data link layers.

intermediate nodes usually only involve the first three layers of TCP/IP (although many of them still have all five layers for other reasons, as we have seen).

In TCP/IP, as with most layered protocols, the most fundamental elements of the process of sending and receiving data are collected into the groups that become the layers. Each layer's major functions are distinct from all the others, but layers can be combined for performance reasons. Each implemented layer has an *interface* with the layers above and below it (except for the application and physical layers, of course) and provides its defined service to the layer above and obtains services from the layer below. In other words, there is a *service interface* between each layer, but these are not standardized and vary widely by operating system.

TCP/IP is designed to be comprehensive and flexible. It can be extended to meet new requirements, and has been. Individual layers can be combined for implementation purposes, as long as the service interfaces to the layers remain intact. Layers can even be split when necessary, and new service interfaces defined. Services are provided to the layer above after the higher layer provides the lower layer with the command, data, and necessary parameters for the lower layer to carry out the task.

Layers on the same system provide and obtain services to and from adjacent layers. However, a *peer-to-peer protocol process* allows the same layers on different systems to communicate. The term *peer* means every implementation of some layer is essentially equal to all others. There is no "master" system at the protocol level. Communications between peer layers on different systems use the defined protocols appropriate to the given layer.

In other words, *services* refer to communications between layers within the same process, and *protocols* refer to communications between processes. This can be confusing, so more information about these points is a good idea.

Protocols and Interfaces

It is important to note that when the layers of TCP/IP are on different systems, they are *only* connected at the physical layer. Direct peer-to-peer communication between all other layers is impossible. This means that all data from an application have to flow "down" through all five layers at the sender, and "up" all five layers at the receiver to reach the correct process on the other system. These data are sometimes called a *service data unit* (SDU).

Each layer on the sending system adds information to the data it receives from the layer above and passes it all to the layer below (except for the physical layer, which has no lower layers to rely on in the model and actually has to send the bits in a form appropriate for the communications link used).

Likewise, each layer on the receiving system unwraps the received message, often called a *protocol data unit* (PDU), with each layer examining, using, and stripping off the information it needs to complete its task, and passing the remainder up to the next layer (except for the application layer, which passes what's left off to the application program itself). For example, the data link layer removes the wrapper meant for it, uses it to decide what it should do with this data unit, and then passes the remainder up to the network layer.

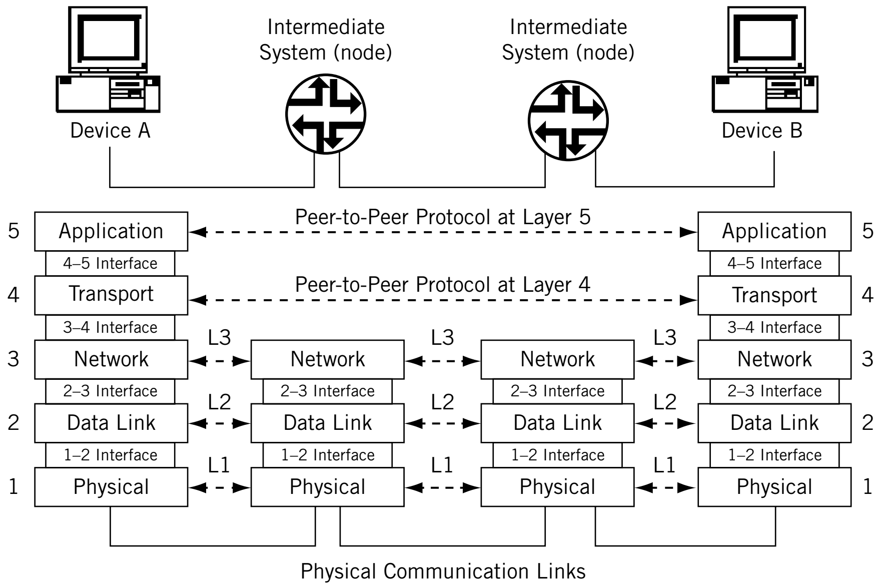


FIGURE 1.10

Protocols and interfaces, showing how devices are only physically connected at the lowest layer (Layer 1). Note that functionally, intermediate nodes only require the bottom three layers of the model.

The whole interface and protocol process is shown in Figure 1.10. Although TCP/IP layers only have names, layer numbers are also used in the figure, but only for illustration. (The numbers come from the ISO-RM.)

As shown in the figure, there is a natural grouping of the five-layer protocol stack at the network layer and the transport layer. The lower three layers of TCP/IP, sometimes called the network support layers, must be present and functional on all systems, regardless of the end system or intermediate node role. The transport layer links the upper and lower layers together. This layer can be used to make sure that what was sent was received, and what was sent is useful to the receiver (and not, for example, a stray PDU misdirected to the host or unreasonably delayed).

The process of encapsulation makes the whole architecture workable. Encapsulation of one layer's information inside another layer is a key part of how TCP/IP works.

Encapsulation

Each layer uses encapsulation to add the information its peer needs on the receiving system. The network layer adds a header to the information it receives from the transport at the sender and passes the whole unit down to the data link layer. At the receiver, the network layer looks at the control information, usually in a *header*, in the data it receives from the data link layer and passes the remainder up to the transport layer for further processing. This is called encapsulation because one layer has no idea what the structure or meaning of the PDU is at other layers. The PDU has several more or less official names for the structure at each layer.

The exception to this general rule is the data link layer, which adds both a *header* and a *trailer* to the data it receives from the network layer. The general flow of encapsulation in TCP/IP is shown in Figure 1.11. Note that on the transmission media itself (or communications link), there are only bits, and that some "extra" bits are added by the communication link for its own purposes. Each PDU at the other layers is labeled as data for its layer, and the headers are abbreviated by layer name. The exception is the second layer, the data link layer, which shows a header and trailer added at that level of encapsulation.

Although the intermediate nodes are not shown, these network devices will only process the data (at most) through the first three layers. In other words, there is no transport layer to which to pass network-layer PDUs on these systems for data communications (management is another issue).

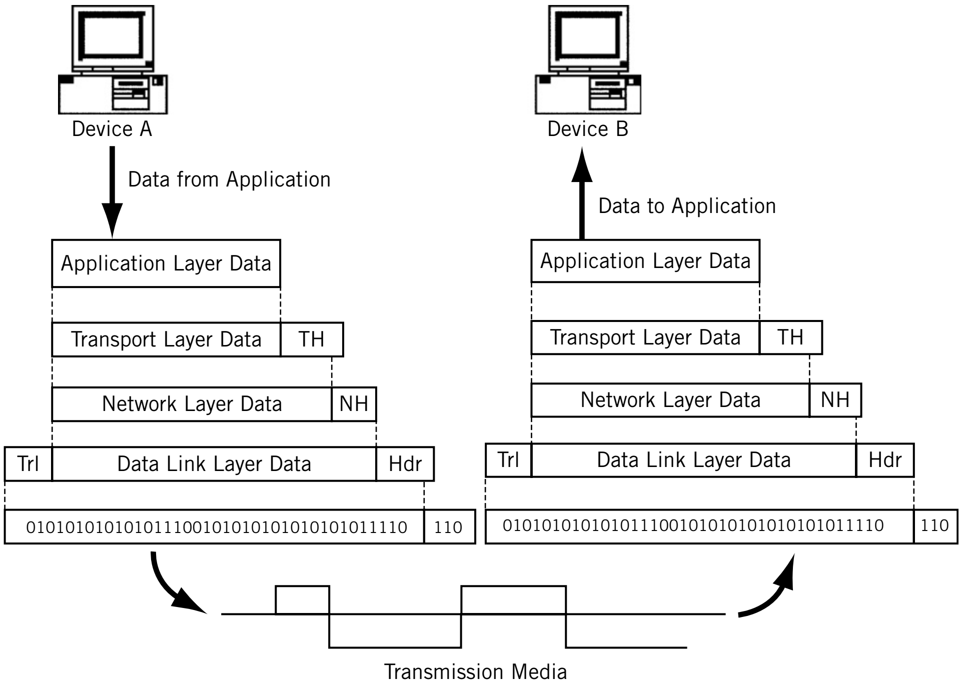


FIGURE 1.11

TCP/IP encapsulation and headers. The unstructured stream of bits represents frames with distinct content.

THE LAYERS OF TCP/IP

TCP/IP is mature and stable, and is the only protocol stack used on the Internet. This book is all about networking with TCP/IP, but it is easy to get lost in the particulars of TCP/IP if some discussion of the general tasks that TCP/IP is intended to accomplish is not included. This section takes a closer look at the TCP/IP layers, but only as a general guide to how the layers work.

TCP/IP Layers in Brief

- **Physical Layer:** Contains all the functions needed to carry the bit stream over a physical medium to another system.
- **Data Link Layer:** Organizes the bit stream into a data unit called a "frame" and delivers the frame to an adjacent system.
- **Network Layer:** Delivers data in the form of a packet from source to destination, across as many links as necessary, to non-adjacent systems.
- **Transport Layer:** Concerned with process-to-process delivery of information.
- **Application Layer:** Concerned with differences in internal representation, user interfaces, and anything else that the user requires.

The Physical Layer

The physical layer contains all the functions needed to carry the bit stream over a physical medium to another system. Figure 1.12 shows the position of the physical layer to the data link layer and the transmission medium. The transmission medium forms a pure "bit pipe" and should not change the bits sent in any way. Now, transmission "on the wire" might send bits through an extremely complex transform, but the goal is to enable the receiver to reconstruct the bit stream exactly as sent. Some information in the form of *transmission framing* can be added to the data link layer data, but this is only used by the physical layer and the transmission medium itself. In some cases, the transmission medium sends a constant idle bit pattern until interrupted by data.

Physical layer specifications have four parts: mechanical, electrical or optical, functional, and procedural. The mechanical part specifies the physical size and shape of the connector itself so that components will plug into each other easily. The electrical/optical specification determines what value of voltage or line condition determines whether a pin is active or what exactly represents a 0 or 1 bit. The functional specification specifies the function of each pin or lead on the connector (first lead is send, second is receive, and so on). The procedural specification details the sequence of actions that must take place to send or receive bits on the interface. (For Ethernet, the send pair is activated, then a "preamble" is sent, and so forth.) The Ethernet twisted-pair interfaces from the IEEE are common implementations of the physical layer that includes all these elements.

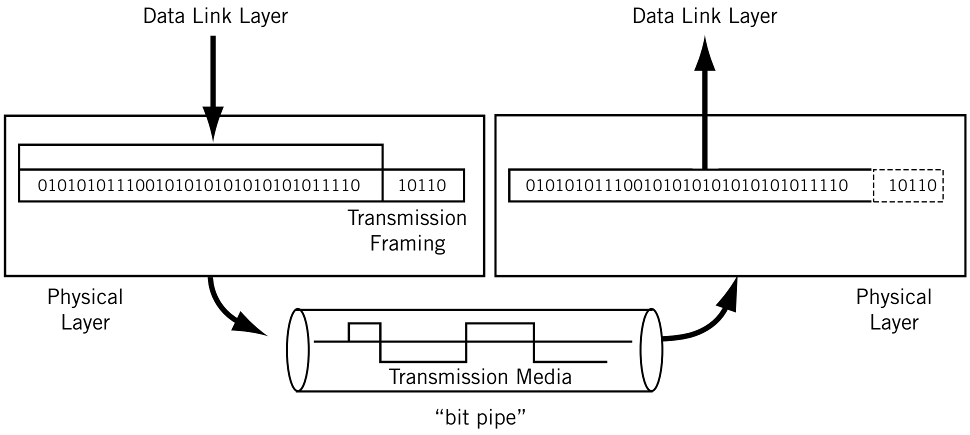


FIGURE 1.12

The physical layer. The transmission framing bits are used for transmission media purposes only, such as low-level control.

There are other things that the physical layer must determine, or be configured to expect.

Data rate—This transmission rate is the number of bits per second that can be sent. It also defines the duration of a symbol on the wire. Symbols usually represent one or more bits, although there are schemes in which one bit is represented by multiple symbols.

Bit synchronization—The sender and receiver must be *synchronized* at the symbol level so that the number of bits expected per unit time is the same. In other words, the sender and receiver *clocks* must be synchronized (timing is in the millisecond or microsecond range). On modern links, the timing information is often "recovered" from the received data stream.

Configuration—So far we've assumed simple *point-to-point links*, but this is not the only way that systems are connected. In a *multipoint configuration*, a link connects more than two devices, and in a multisystem *bus/broadcast* topology such as a LAN, the number of systems can be very high.

Topology—The devices can be arranged in a number of ways. In a full *mesh* topology, all devices are directly connected and *one hop* away, but this requires a staggering amount of links for even a modest network. Systems can also be arranged as a *star* topology, with all systems reachable through a central system. There is also the *bus* (all devices are on a common link) and the *ring* (devices are chained together, and the last is linked to the first, forming a ring).

Mode—So far, we've only talked about one of the systems as the sender and the other as the receiver. This is operation in *simplex mode*, where a device can only send or receive, such as with weather sensors reporting to a remote weather station. More realistic devices use *duplex mode*, where all systems can send or receive with equal facility. This is often further distinguished as *half-duplex* (the system can send and receive, but not at the same time) and *full-duplex* (simultaneous sending and receiving).

The Data Link Layer

Bits are just bits. With only a physical layer, System A has no way to tell System B, "Get ready some bits," "Here are the bits," and "Did you get those bits okay?" The data link layer solves this problem by organizing the bit stream into a data unit called a frame.

It is important to note that frames are the data link layer PDUs, and these are not the same as the physical layer transmission frames mentioned in the previous section. For example, network engineers often speak about *T1 frames* or *SONET frames*, but these are distinct from the data link layer frames that are carried inside the T1 or SONET frames. Transmission frames have control information used to manage the physical link itself and has little to do directly with process-to-process communications. This "double-frame" arrangement might sound redundant, but many transmission frames originated with voice because digitized voice has no framing at the "data link" layer.

The data link layer moves bits across the link and can add reliability to the raw communications link. The data link layer can be very simple, or make the link appear error-free to the layer above, the network layer. The data link layer usually adds both a header and trailer to the data presented by the network layer. This is shown in Figure 1.13.

The frame header typically contains a source and destination address (known as the "physical address" since it refers to the physical communication port) and some control information. The control information is data passed from one data link layer to the

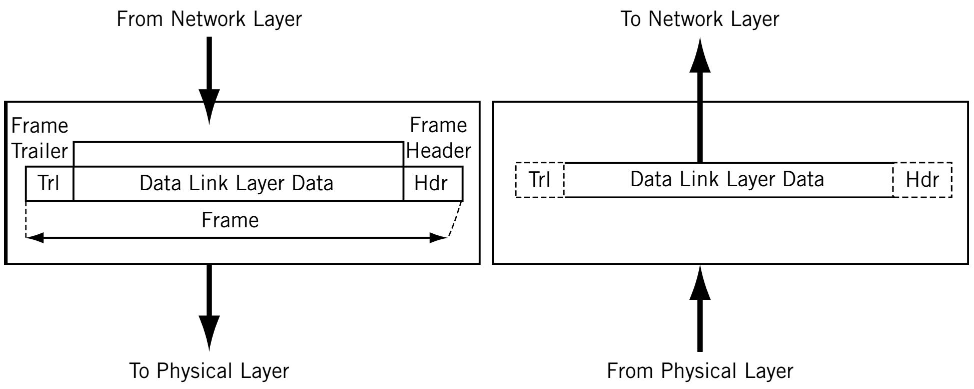


FIGURE 1.13

The data link layer, showing that data link layer frames have both header and trailer.

other data link layer, and not user data. The body of the frame contains the sequence of bits being transferred across the network. The trailer usually contains information used in detecting bit errors (such as cyclical redundancy check [CRC]). A maximum size is associated with the frame that cannot be exceeded because all systems must allocate memory space (buffers) for the data. In a networking context, a buffer is just special memory allocated for communications.

The data link layer performs framing, physical addressing, and error detection (error *correction* is another matter entirely, and can be handled in many ways, such as by resending a copy of the frame that had the errors). However, when it comes to frame error detection and correction in the real world, error detection bits are sometimes ignored and frames that defy processing due to errors are simply discarded. This does not mean that error detection and correction are not part of the data link layer standards: It means that in these cases, ignoring and discarding are the chosen methods of implementation. In discard cases, the chore of handling the error condition is "pushed up the stack" to a higher layer protocol.

This layer also performs *access control* (this determines whose turn it is to send over or control the link, an issue that becomes more and more interesting as the number of devices sharing the link grows). In LANs, this *media access control* (MAC) forms a sublayer of the data link layer and has its own addressing scheme known (not surprisingly) as the MAC layer address or *MAC address*. We'll look at MAC addresses in the next chapter. For now, it is enough to note that LANs such as Ethernet do not have "real" physical layer addresses and that the MAC address performs this addressing function.

In addition, the data link layer can perform some type of *flow control*. Flow control makes sure senders do not overwhelm receivers: a receiver must have adequate time to process the data arriving in its buffers. At this layer, the flow control, if provided, is link-by-link. (We'll see shortly that end-to-end—host-to-host—flow control is provided by the transport layer.) LANs do not usually provide flow control at the data link layer, although they can.

Not all destination systems are directly reachable by the sender. This means that when bits at the data link layer are sent from an originating system, the bits do not arrive at the destination system as the "next hop" along the way. Directly reachable systems are called *adjacent systems*, and adjacent systems are always "one hop away" from the sender. When the destination system is not directly reachable by the sender, one or more intermediate nodes are needed. Consider the network shown in Figure 1.14.

Now the sender (System A) is not directly connected to the receiver (System B). Another system, System 3, receives the frame and must forward it toward the destination. This system is usually called a *switch* or *router* (there are even other names), depending on internal architecture and network role. On a WAN (but not on a LAN), this second frame is a different frame because there is no guarantee that the second link is identical to the first. Different links need different frames. Identical frames are only delivered to systems that are directly reachable, or adjacent, to the sender, such as by an Ethernet switch on a LAN.

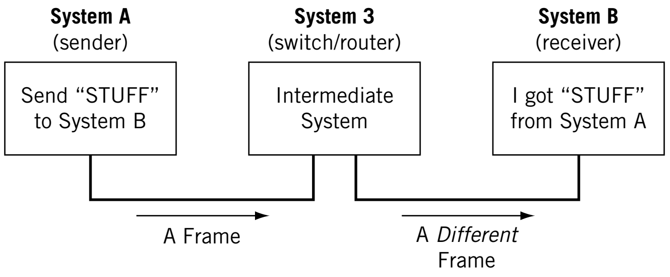


FIGURE 1.14

A more complex network. Note that the frames are technically different even if the same medium is used on both links.

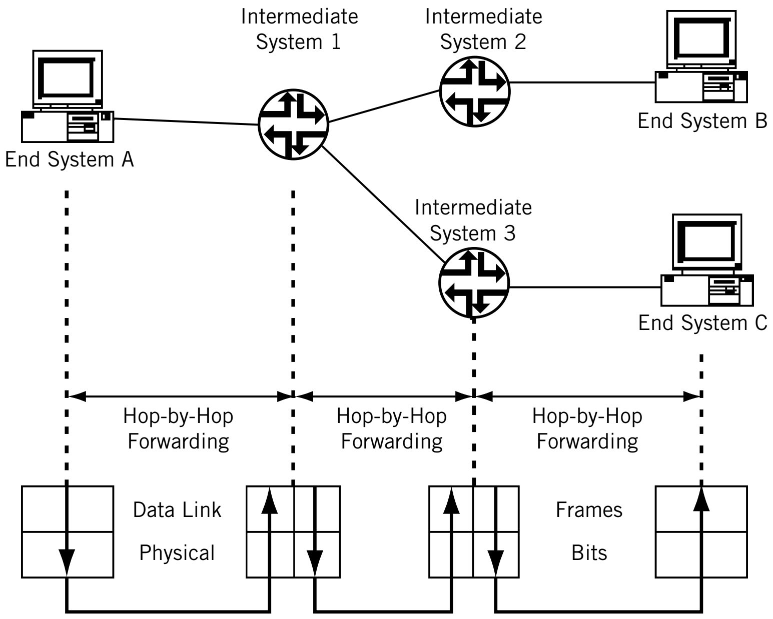


FIGURE 1.15

Hop-by-hop forwarding of frames. The intermediate systems also have a Layer 3, but this is not shown in the figure for clarity.

Networking with intermediate systems is called *hop-by-hop* delivery. A "hop" is the usual term used on the Internet or a router network to indicate the forwarding of a packet between one router or another (or between a host and router). Frames can "hop" between Layer 2 switches, but the term is most commonly used for Layer 3 router hops (which can consist of multiple switch-to-switch frame "hops"). There can be more than one intermediate system between the source and destination end systems, of course, as shown in Figure 1.15. Consider the case where End System A is sending a bit stream to End System C.

Note that the intermediate systems (routers) have *two* distinct physical and data link layers, reflecting the fact that the systems have two (and often more) communication links, which can differ in many ways. (The figure shows a typical WAN configuration with point-to-point links, but routers on LANs, and on some types of public data service WANs, can be deployed in more complicated ways.)

However, there is something obviously missing from this figure. There is no connection between the data link layers on the intermediate systems! How does the router know to which output port and link to forward the data in order to ultimately reach the destination? (In the figure, note that Intermediate System 1 can send data to either Intermediate System 2 or Intermediate System 3, but only through Intermediate System 3, which forwards the data, is the destination reachable.)

These forwarding decisions are made at the TCP/IP network layer.

The Network Layer

The network layer delivers data in the form of a *packet* from source to destination, across as many links as necessary. The biggest difference between the network layer and the data link layer is that the data link layer is in charge of data delivery between *adjacent* systems (directly connected systems one hop away), while the network layer delivers data to systems that are not directly connected to the source. There can be many different types of data link and physical layers on the network, depending on the variety of the link types, but the network layer is essentially the same on all systems, end systems, and intermediate systems alike.

Figure 1.16 shows the relationship between the network layer and the transport layer above and the data link layer below. A packet header is put in place at the sender and interpreted by the receiver. A router simply looks at the packet header and makes a forwarding decision based on this information. The transport layer does not play a role in the forwarding decision.

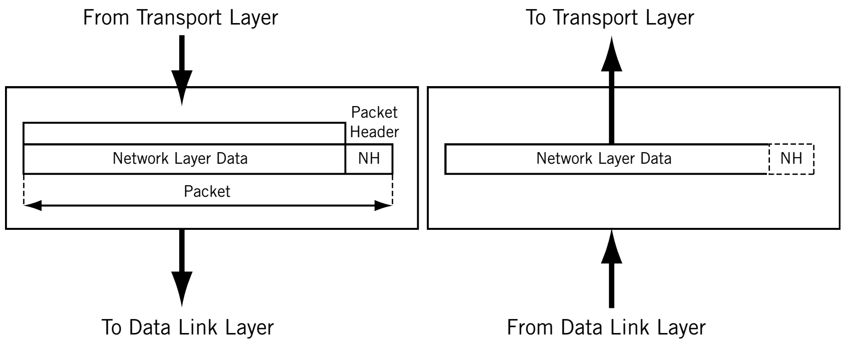


FIGURE 1.16

The network layer. These data units are packets with their own destination and source address formats.

How does the network layer know where the packet came from (so the sender can reply)? The key concept at the network layer is the *network address*, which provides this information. In TCP/IP, the network address is the *IP address*.

Every system in the network receives a network address, whether an end system or intermediate system. Systems require at least one network address (and sometimes many more). It is important to realize that this network address is different from, and independent of, the physical address used by the frames that carry the packets between adjacent systems.

Why should the systems need two addresses for the two layers? Why can't they just both use either the data link ("physical") address or the network address at *both* layers? There are actually several reasons. First, LAN addresses like those used in Ethernet come from one group (the IEEE), while those used in TCP/IP come from another group (ICANN). Also, the IP address is universally used on the Internet, while there are many types of physical addresses. Finally, there is no systematic assignment of physical addresses (and many addresses on WANs can be duplicates and so have "local significance only"). On the other hand, IP network addresses are globally administered, unique, and have a portion under which many devices are grouped. Therefore, many devices can be addressed concisely by this *network portion* of the IP address.

A key issue is how the network addresses "map" to physical addresses, a process known generally as *address resolution*. In TCP/IP, a special family of address resolution protocols takes care of this process.

The network address is a logical address. Network addresses should be organized so that devices can be grouped under a part of that address. In other words, the network address should be organized in a fashion similar to a telephone number, for example, 212-555-1212 in the North American public switched telephone network (PSTN). The sender need only look at the area code or "network" portion of this address (212) to determine if the destination is local (area codes are the same) or needs to be sent to an intermediate system to reach the 212 area code (source and destination area codes differ).

For this scheme to work effectively, however, all telephones that share the 212 area code should be grouped together. The whole telephone number beginning with 212 therefore means "*this* telephone in the 212 area code." In TCP/IP, the network address is the beginning of the device's complete IP address. A group of hosts is gathered under the network portion of the IP address. IP network addresses, like area codes, are globally administered to prevent duplication, while the rest of the IP address, like the rest of the telephone number, is locally administered, often independently.

In some cases, the packet that arrives at an intermediate system inside a frame is too large to fit inside the frame that must be sent out. This is not uncommon: different link and LAN types have different maximum frame sizes. The network layer must be able to *fragment* a data unit across multiple frames and reassemble the fragments at the destination. We'll say more about *fragmentation* in a later chapter.

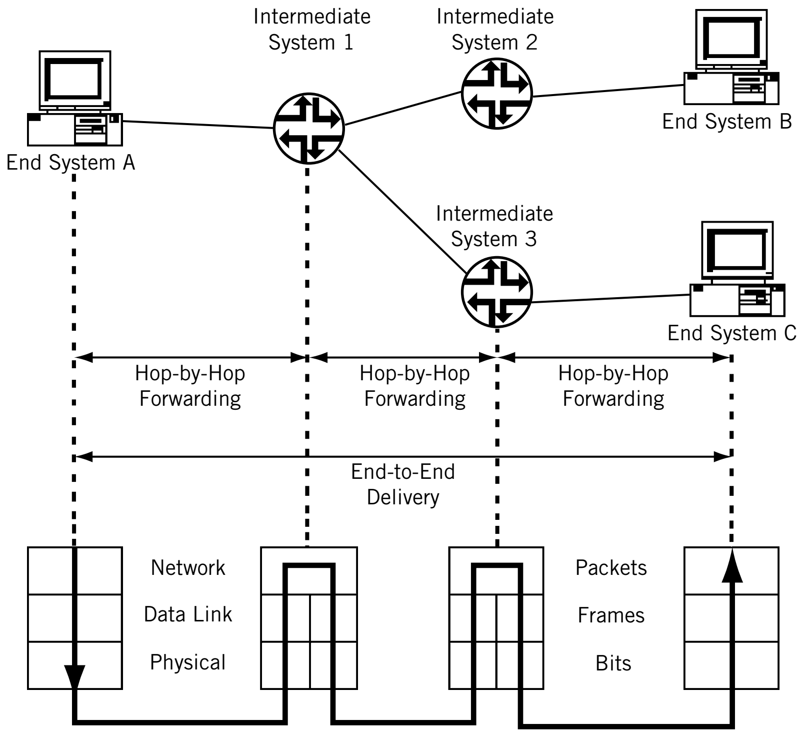


FIGURE 1.17

Source-to-destination delivery at the network layer. The intermediate systems now have all three required layers.

The network layer uses one or more *routing tables* to store information about reachable systems. The routing tables must be created, maintained, and purged of old information as the network changes due to failures, the addition or deletion of systems and links, or other configuration changes. This whole process of building tables to pass data from source to destination is called *routing*, and the use of these tables for packet delivery is called *forwarding*. The forwarding of packets inside frames always takes place hop by hop. This is shown in Figure 1.17, which adds the network layer to the data link layers already present and distinguishes between hop-by-hop forwarding and end-to-end delivery.

On the Internet, the intermediate systems that act at the packet level (Layer 3) are called *routers*. Devices that act on frames (Layer 2) are called *switches*, and some older telephony-based WAN architectures use switches as intermediate network nodes. Whether a node is called a switch or router depends on how they function internally.

In a very real sense, the network layer is at the very heart of any protocol stack, and TCP/IP is no exception. The protocol at this layer is IP, either IPv4 or IPv6 (some think that IPv6 is distinct enough to be known as TCPv6/IPv6).

The Transport Layer

Process-to-process delivery is the task of the transport layer. Getting a packet to the destination system is not quite the same thing as determining which process should receive the packet's *content*. A system can be running file transfer, email, and other network processes all at the same time, and all over a single physical interface. Naturally, the destination process has to know on which process the sender originated the bits inside the packet in order to reply. Also, systems cannot simply transfer a huge multimegabit file all in one packet. Many data units exceed the maximum allowable size of a packet.

This process of dividing message content into packets is known as *segmentation*. The network layer forwards each and every packet independently, and does not recognize any relationship between the packets. (Is this a file transfer or email packet? The network layer does not care.) The transport layer, in contrast, can make sure the whole *message*, often strung out in a sequence of packets, arrives in order (packets can be delivered out of sequence) and intact (there are no errors in the entire message). This function of the transport layer involves some method of flow control and error control (error detection *and* error correction) at the transport layer, functions which are absent at the network layer. The transport-layer protocol that performs all of these functions is TCP.

The transport-layer protocol does not have to do any of this, of course. In many cases, the content of the packet forms a complete unit all by itself, called a *datagram*. (The term "datagram" is often used to refer to the whole IP packet, but not in this book.) Self-contained datagrams are not concerned with sequencing or flow control, and these functions are absent in the User Datagram Protocol (UDP) at the transport layer.

So there are two very popular protocol packages at the transport layer:

- *TCP*—This is a connection-oriented, "reliable" service that provides ordered delivery of packet contents.
- *UDP*—This is a connectionless, "unreliable" service that does *not* provide ordered delivery of packet contents.

In addition to UDP and TCP, there are other transport-layer protocols that can be used in TCP/IP, all of which differ in terms of how they handle transport-layer tasks. Developers are not limited to the standard choices for applications. If neither TCP nor UDP nor any other defined transport-layer service is appropriate for your application, you can write your own transport-layer protocols and get others to adapt it (or use your application package exclusively).

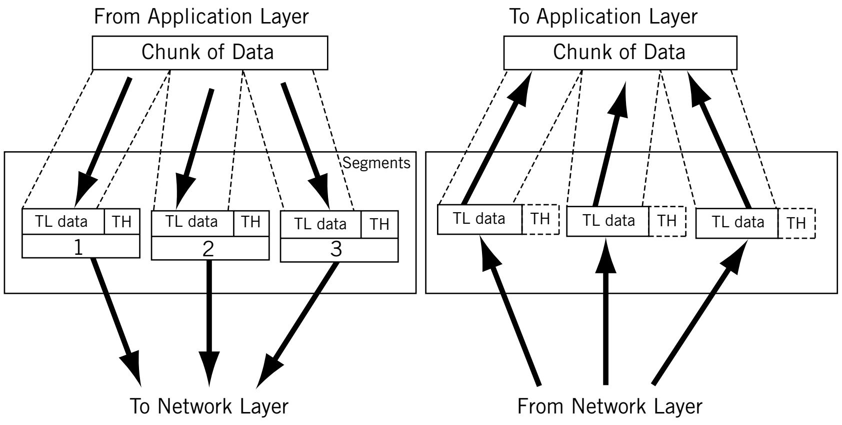


FIGURE 1.18

The transport layer, showing how data are broken up if necessary and reassembled at the destination.

In TCP/IP, it is often said that the network layer (IP itself) offers an "unreliable" or "best effort" service, while the transport layer adds "reliability" in the form of flow and error control. Later in this book, we'll see why these terms are unfortunate and what they really mean.

The network layer gets a single packet to the right system, and the transport layer gets the entire message to the right process. Figure 1.18 shows the transport layer breaking up a message at the sender into three pieces (each labeled "TL data" for transport-layer data and "TH" for transport-layer header). The figure then shows the transport layer reassembling the message at the receiver from the various *segments* that make up a message. In TCP/IP, there are also data units known as *datagrams*, which are always handled as self-contained units. There are profound differences between how the transport layer treats segments and datagrams, but this figure is just a general illustration of segment handling.

The functions that the transport layer, which in some protocols is called the end-to-end layer, might have to include follow:

Process addressing and multiplexing—Also known as "service-point addressing," the transport layer has to decide which process originated the message and to which process the message must be delivered. These are also known as *port addresses* in TCP/IP. Port addresses are an important portion of the application *socket* in TCP/IP.

Segment handling—In cases where each message is divided into segments, each segment has a sequence number used to put the message back together at the destination. When datagrams are used, each data unit is handled independently and sequencing is not necessary.

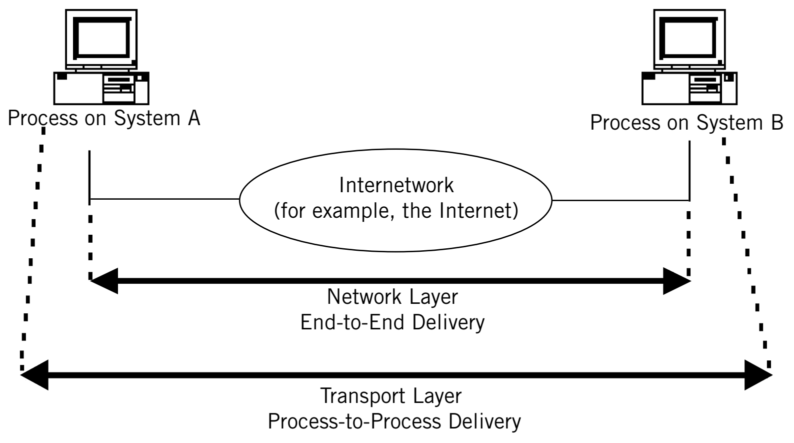


FIGURE 1.19

Reliable process-to-process delivery with the transport layer.

Connection control—The transport layer can be *connectionless* or *connection-oriented* (in fact, several layers can operate in either one of these ways). Connectionless (CL) layers treat every data unit as a self-contained, independent unit. Connection-oriented (CO) layers go through a three-phase process every time there is data to send to a destination after an idle period (connection durations can vary). First, some control messages establish the connection, then the data are sent (and exchanged if replies are necessary), and finally the connection is closed. Many times, a comparison is made between a telephone conversation ("dial, talk, hang up") with connections and an intercom ("push and talk any time") for connectionless communications, but this is not precise. Generally, segments are connection-oriented data units, and datagrams are connectionless data units.

Flow control—Just as with the data link layer, the transport layer can include flow control mechanisms to prevent senders from overwhelming receivers. In this case, however, the flow control is end-to-end rather than link-by-link. Datagrams do not require this service.

Error control—This is another function that can be performed at the data link layer, but again end-to-end at the transport layer rather than link-by-link. Communications links are not the only source of errors, which can occur inside a system as well. Again, datagrams do not require this service.

Figure 1.19 shows the relationship between the network layer and transport layer more clearly. The network layer operates from network interface to network interface, while the transport layer is more specific and operates from process to process.

The Application Layer

It might seem that once data are transferred from end-system process to end-system process, the networking task is pretty much complete. There is a lot that still needs to be done at the application level itself. In models of protocol stacks, it is common to place another layer between the transport layer and the user, the application layer. However, the TCP/IP protocol stack really stops at the transport layer (where TCP and UDP are). It is up to the application programmer to decide what should happen at the client and server level at that point, although there are individual RFCs for guidance, such as for FTP.

Although it is common to gather these TCP/IP applications into their own layer, there really is no such thing in TCP/IP as an application layer to act as some kind of "glue" between the application's user and the network.

In nearly all TCP/IP stacks, the application layer is part of the application process. In spite of the lack of a defined layer, a TCP/IP application might still have a lot to do, and in some ways the application layer is the most complex "layer" of all.

There are two major tasks that the application often needs to accomplish: session support and conversion of internal representation. Not all applications need both, of course, and some applications might not need either, but this overview includes both major functions.

Session Support

A session is a type of *dialog controller* between two processes that establishes, maintains, and synchronizes (controls) the interaction (dialog). A session decides if the communication can be half-duplex (both ends take turns sending) or full-duplex (both ends can send whenever they want). It also keeps a kind of "history" of the interaction between endpoints, so that when things go wrong or when the two communicate again, some information does not have to be resent.

In practical terms, the session consists of all "state variables" necessary to construct the history of the connection between the two devices. It is more difficult, but not impossible, to implement sessions in a connectionless environment because there is no easy way to associate the variables with a convenient label.

Internal Representation Conversion

The role of internal representation conversion is to make sure that the data exchange over the network is useful to the receivers. If the internal representation of data differs on the two systems (integer size, bit order in memory, etc.), the application layer translates between the formats so the application program does not have to. This layer can also provide encryption and compression functions, although it is more common to implement these last two functions separately from the network.

Standard protocol specifications can use the Abstract Syntax Notation 1 (ASN.1) definitions for translation purposes. ASN.1 can be used in programming, network

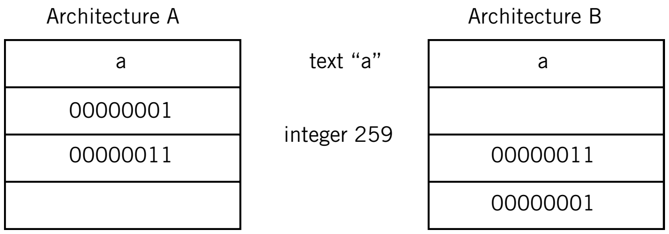


FIGURE 1.20

Internal representation differences. Integers can have different bit lengths and can be stored differently in memory.

management, and other places. ASN.1 defines various things such as which bit is "first on the wire" regardless of how it is stored internally, how many bits are to be sent for the numbers 0 through 255 (8), and so on. Everything can be translated into ASN.1, sent across the network, and translated back to whatever internal format is required at the destination.

The role of internal representation conversion is shown in Figure 1.20. The figure shows four sequential memory locations, each storing the letter "a" followed by the integer 259. Note that not only are there differences between the amount of memory addressed at once, but also in the *order* of the bits for numerics.

In some protocol stacks, the application program can rely on the services of a fully functional conversion for internal representation to perform these services. However, in TCP/IP, every network application program must do these things for itself.

Applications in TCP/IP

TCP/IP does not provide session or presentation services directly to an application. Programmers are on their own, but this does not mean they have to create everything from scratch. For example, applications can use a character-based presentation service called the Network Virtual Terminal (NVT), part of the Internet's telnet remote access specification. Other applications can use Sun's External Data Representation (XDR) or IBM's (and Microsoft's) NetBIOS programming libraries for presentation services. In this respect, there are many presentation layer services that TCP/IP can use, but there is no formal presentation service standard in TCP/IP that all applications must use.

Host TCP/IP implementations typically provide a range of applications that provide users with access to the data handled by the transport-layer protocols. These applications use a number of protocols that are not part of TCP/IP proper, but are used with TCP/IP. These protocols include the Hyper-Text Transfer Protocol (HTTP) used by Web browsers, the Simple Message Transfer Protocol (SMTP) used for email, and many others.

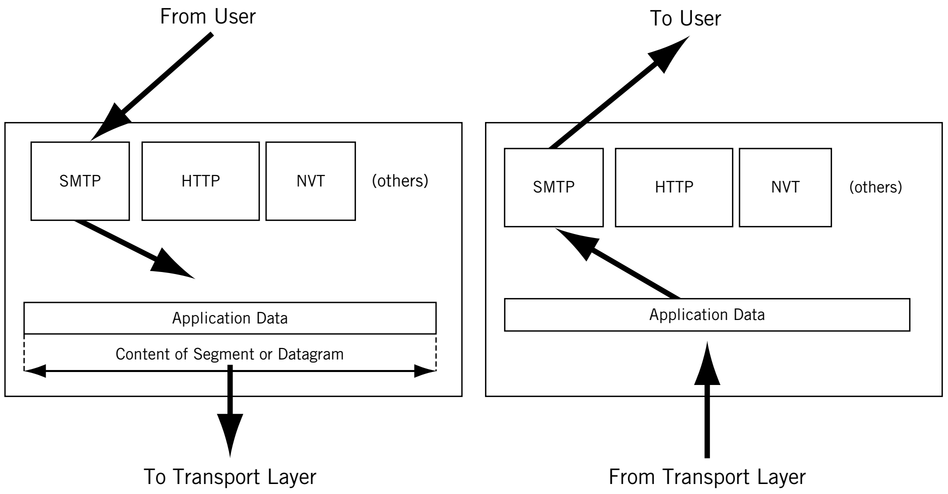


FIGURE 1.21

TCP/IP applications, showing how multiple applications can all share the same network connection.

In TCP/IP, the application protocol, the application service, and the user application itself often share the same name. The file transfer protocol in TCP/IP, called FTP, is at once an application protocol, an application service, and an application run by a user. It can sometimes be confusing as to just which aspect of FTP is under discussion.

The role of TCP/IP applications is shown in Figure 1.21. Note that this "layer" sits on top of the TCP/IP protocol stack and interfaces with programs or users directly.

Some protocols provide separate layers for sessions, internal representation conversion, and application services. In practice, these are seldom implemented independently. It just makes more sense to bundle them together by major application, as in TCP/IP.

THE TCP/IP PROTOCOL SUITE

To sum up, the five layers of TCP/IP are physical, data link, network, transport, and application. The TCP/IP stack is a hierarchical model made up of interactive modules. Each module provides a specific function. In TCP/IP, the layers contain relatively independent protocols that can be "mixed and matched" depending on the needs of the system to provide whatever function is desired. TCP/IP is hierarchical in the sense that each higher layer protocol is supported by one or more lower layer protocols.

Figure 1.22 maps some of the protocols used in TCP/IP to the various layers of TCP/IP. Every protocol in the figure will be discussed in this book, most in chapters all their own.

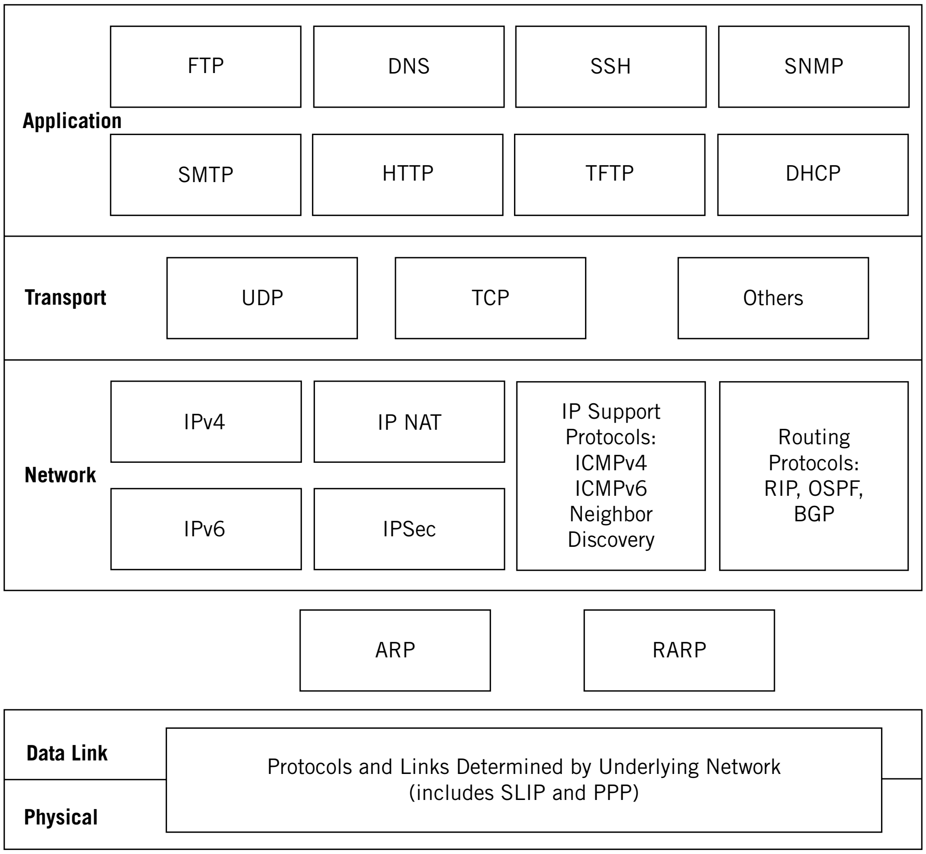

Application
FTP
DNS
SSH
SNMP
SMTP
HTTP
TFTP
DHCP
Transport
UDP
TCP
Others
Network
IPv4
IP NAT
IP Support Protocols: ICMPv4 ICMPv6 Neighbor Discovery
Routing Protocols: RIP, OSPF, BGP
IPv6
IPSec
ARP
RARP
Data Link
Physical
Protocols and Links Determined by Underlying Network (includes SLIP and PPP)

FIGURE 1.22

TCP/IP protocols and layers. Note the position of some protocols between layers.

With few exceptions, the TCP/IP protocol suite does not really define any low-level protocols below the network layer. TCP/IP usually specifies how to put IP packets into frames and how to get them out again. Many RFCs define IP *mapping* into these lower-layer protocols. We'll talk more about this mapping process in Chapter 2.

QUESTIONS FOR READERS

Refer to Figure 1.23 to help you answer the following questions.

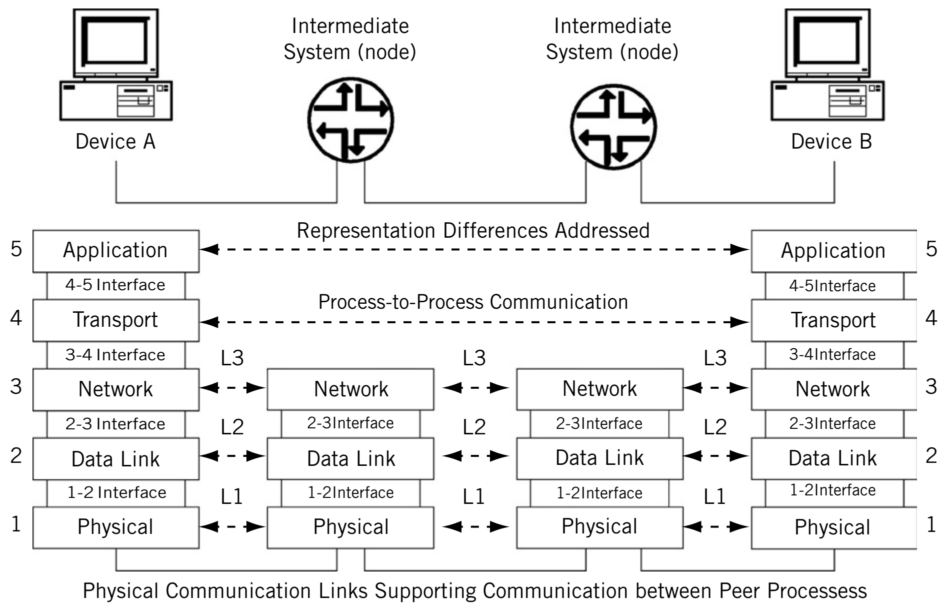

Intermediate System (node)
Intermediate System (node)
Device A
Device B
Representation Differences Addressed
Process-to-Process Communication
5 Application
4-5 Interface
4 Transport
3-4 Interface
3 Network
2-3 Interface
2 Data Link
1-2 Interface
1 Physical
L3
L2
L1
Physical Communication Links Supporting Communication between Peer Processess

FIGURE 1.23

Summary of layered communications.

1. What are the differences between network-layer delivery and transport-layer delivery?
2. What are the main characteristics of a peer-to-peer process?
3. What are port addresses, logical addresses, and physical addresses?
4. What are the functions of the data link layer in the Internet model?
5. Which two major types of services can be provided at the application "layer"?