Copyright

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers discounts on this book when ordered in quantity for special sales. For more information, please contact:

Pearson Education Corporate Sales Division

201 W. 103rd Street

Indianapolis, IN 46290

(800) 428-5331

corpsales@pearsontechgroup.com

Visit AW on the Web: www.awprofessional.com

Library of Congress Control Number: 2002106713

Copyright © 2002 by Pearson Education, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher. Printed in the United States of America. Published simultaneously in Canada.

For information on obtaining permission for use of material from this work, please submit a written request to:

Pearson Education, Inc.

Rights and Contracts Department

75 Arlington Street, Suite 300

Boston, MA 02116

Fax: (617) 848-7047

Text printed on recycled paper

1 2 3 4 5 6 7 8 9 10—MA—0605040302

First printing, June 2002

Dedication

To all those who donated time, money, products, or blood after September 11, 2001. You give hope that humanity's best side may win out over its worst side.

Preface

Computer networks have changed our lives. They grew slowly, and mostly unnoticed, in the 1970s and 1980s. In the 1990s, though, something happened. Perhaps it was the availability of the World Wide Web (WWW, or Web) and graphical Web browsers, which made computer networks accessible to Grandma Dorothy and Uncle Stan. Maybe it was that the availability of network connections had reached a critical threshold. Perhaps the quality and quantity of network-enabled software passed a critical threshold. Possibly it was two or all three of these things, or something else entirely. In any event, networks became noticeable. Most importantly, *the Internet* became noticeable.

The Internet comprises millions of computers, many of which run *servers*—software packages designed to listen for and respond to data transfer requests from other computers. Because the protocols upon which the Internet was built were designed to work in a cross-platform manner, both Internet clients and the servers they use run on many different platforms. One of the most popular of these is Linux. Coupled with inexpensive *x*86 hardware, Linux makes a very cost-effective server platform for small and mid-sized sites. Indeed, with increasing computer performance and Linux versions working their way up the computer performance hierarchy, Linux is beginning to make inroads into the large server market. Thus, with Linux on everything from tiny network appliances to large servers, knowing how to set up and maintain a Linux server system is an important skill for networking professionals today.

Which servers, though? There are hundreds, if not thousands, of individual server programs. Most general-purpose Linux networking books focus on a handful of popular servers—Web (HTTP) servers like Apache, login servers like Telnet and SSH, file servers like NFS and Samba, and a few others. These books present enough information to get a user up and running, but little more. They also give short shrift to servers that are less visible but that are often extremely important, like DHCP servers, time servers, and Kerberos. This book takes a different approach to Linux networking: I assume that you know at least a minimal amount about Linux and networking in general, and you want to take you skills to a higher level. Although this book does cover the "usual suspects," it spends less time introducing the basics and more time describing advanced or unusual configurations. This book also covers some of the servers and topics that are neglected in most entry-level Linux networking books. The result is the closest thing possible to a book that's both a *general* Linux networking book and an *advanced* Linux networking book.

To be sure, you won't learn everything there is to know about complex packages like Apache or Samba in this book. The relevant chapters provide quick introductions to these tools, a summary of some popular techniques you won't find covered in other introductory Linux networking books, and pointers to additional resources. This book's approach is to be a general-purpose Linux networking book for people who are not novices.

Who Should Buy This Book

This book is designed to be an advanced tutorial and reference for those with some Linux networking experience, or at least some Linux and some networking experience. The first few chapters cover low-level configuration, including such factors as getting the network up and running to begin with; but I assume you're already familiar with Linux, or at least UNIX, and with basic networking terminology. If you're not familiar with these things, an introductory Linux system administration book, such as Marcel Gagné's *Linux System Administration: A User's Guide* (Addison-Wesley, 2002) or Vicki Stanfield's and my *Linux System Administration* (Sybex, 2001) should help fill in the gaps.

If you want to learn a bit more about big servers like Apache or Samba but don't want to buy dedicated books for them, or if you want to learn about the small but potentially important servers like xntpd or xfs, then this is the book for you. This book also covers miscellaneous networking topics, like how to start and stop servers, backing up a network, running a server in a chroot jail, and using iptables. Knowing these topics will help fill out your networking knowledge base and make you better able to adapt to new requirements and generally improve the networks you administer.

In writing this book, I imagined the audience to be administrators of small or mid-sized networks. Your network might be dominated by Linux, UNIX, Windows, MacOS, or something even more exotic, but of course you've got at least one Linux system. Most chapters describe the basic principles upon which a tool is built and then describe how to use the tool. You should, therefore, be able to learn a lot about the tools by reading this book, but you can also use this book as a quick reference. I aim for this to be the book you would choose if you could have just one Linux networking book.

Linux Distributions

One of the challenges of administering Linux is that Linux isn't a single OS. Instead, it's a collection of OSs, all built around the same kernel. Each of these variant OSs is known as a *distribution.* A distribution consists of a Linux kernel; a distribution-specific installation program; a wide assortment of support tools, user programs, and so on; and a set of default startup and configuration scripts. Different distributions frequently use different versions of the Linux kernel and of support programs. Indeed, they sometimes ship with different programs entirely to fill particular roles, such as sendmail, Exim, or Postfix for a mail server. For these reasons, Linux distributions can vary substantially in overall feel and in many administrative details.

Many books on Linux fail to address the variability among Linux distributions. They intentionally focus on just one distribution, or provide coverage of others in a cursory manner. One of the goals of this book, though, is to cover several of the most popular Linux distributions explicitly. Specifically, I cover Caldera OpenLinux 3.1, Debian GNU/Linux 2.2, Mandrake 8.1, Red Hat 7.2, Slackware 7.0, SuSE 7.3, and TurboLinux 7.0. To be sure, I can't cover every detail for each of these OSs, but I point out where they differ in important ways, such as where each places network startup scripts and what FTP servers each includes. Some chapters—notably those on server startup tools, LPD print servers, SMTP mail servers, and FTP servers—cover multiple servers in order to be applicable to the default configurations for each of these seven major Linux distributions.

How This Book Is Organized

This book is broken down into four parts of four to thirteen chapters. The structure represents the assumption that your network includes some servers that are used primarily by local users and others that are exposed to the Internet at large, but of course some servers can do double duty, so the placement of some servers may not reflect the configuration on your network. The book's four parts are as follows:

- **Part I: Low-Level Configuration**— This part is the shortest, at only four chapters. It covers kernel network configuration options, basic TCP/IP network configuration, network stacks, and starting servers.

- **Part II: Local Network Servers**— This part covers servers and procedures that are most likely to be used by other computers on your local network. This includes DHCP servers, Kerberos, Samba, NFS servers, printing with LPD, time servers, POP and IMAP mail servers, news servers, remote login servers, X and VNC servers, font servers, remote administration servers, and network backups.

- **Part III: Internet Servers**— These chapters cover servers that are often accessible to the world at large. Topics include DNS servers, SMTP mail servers, Web servers, and FTP servers.

- **Part IV: Network Security and Router Functions**— This part diverges from the emphasis on servers as distinct topics for chapters and focuses on network security. Topics include a general look at network security, using a chroot jail, configuring advanced router functions, using iptables for firewall and NAT purposes, and setting up a VPN.

Conventions Used in This Book

In discussing computers and software, it's easy to become confused because it's not always clear when a word has its usual meaning and when it refers to a computer, file, program, command, or what have you. For this reason, this book uses certain typographic conventions to help clarify matters. Specifically:

- The bulk of the text appears in a normal, proportionally-spaced font, like this.
- *Italicized text* indicates an important term that's appearing for the first time in a chapter. It's also occasionally used for emphasis.
- Monospaced text indicates a filename, computer name, the syntax used by a command, the contents of configuration files, or the output of commands typed at a command prompt. Sometimes program names appear in this way, when these names are really the software's filename.
- *Italicized monospaced text* indicates a variable—information that may differ on your system. For instance, instructions might say to create a file whose name is unimportant or system-specific. The instructions might then refer to this file as *file.txt*.
- **Bold monospaced text** indicates information you should type exactly at a command prompt. When isolated on a line of its own, it's usually preceded by a monospaced but non-bold prompt, such as #, which the computer generates. This type of text may also be italicized, to indicate that what you type will depend upon your configuration or the results you intend to achieve.

When a command you type appears on a line of its own, the command is preceded by a command prompt. A pound sign (#) indicates a root command prompt. Such commands are usually entered by root, not by ordinary users (although there are exceptions to this rule). If the command prompt is a dollar sign ($), ordinary users may, and often do, enter the command. Some unusually long commands use line continuation characters—backslashes (\)—at the ends of all their lines but the first. You can type such commands exactly as they appear, including the backslashes, or you can omit the backslashes and type these commands entirely on one line. The backslashes exist just so that the command can be typeset in a reasonable font size.

This book also uses a number of special text elements that apply to entire paragraphs or larger segments of text. These elements are intended to highlight important or peripheral information. They are:

NOTE

A Note is not critical to the main discussion, but the information it

contains is interesting or may be helpful in certain circumstances. For instance, a Note might point out how a feature differed in previous versions of a program.

TIP

A Tip contains information that can help you achieve a goal in a non-obvious way, or that can point you to uses of a system or software that might not have occurred to you.

WARNING

A Warning describes a potential pitfall or danger. Warnings include software that could damage your system if used incorrectly, the potential to run afoul of ISP policies that forbid certain behaviors, and configurations that might leave your system vulnerable to outside intruders.

Sidebars

A Sidebar is like a Note, but it's usually longer—typically at least two paragraphs. These components contain extended discussion of issues that don't fit neatly into the overall flow of the chapter, but that are nonetheless related, interesting, or even important.

In discussing networks, it's often necessary to give specific IP addresses as examples. In most cases, I've used IP addresses from the ranges reserved for private networks (192.168.0.0–192.168.255.255, 172.16.0.0–172.31.255.255, and 10.0.0.0–10.255.255.255) even for systems that would normally be on the Internet at large. I've done this to avoid potential confusion or inadvertent offense that might occur if I were to pick random legitimate IP addresses.

Contacting Me

If you have questions or comments about this book, I can be reached at rodsmith@rodsbooks.com. I also maintain a Web page about the book at http://www.rodsbooks.com/adv-net/.

Acknowledgments

I'd like to thank my editor, Karen Gettman, for her careful work shepherding this book through the production process. She was helped in this task by Emily Frey, the project coordinator, who received my chapters and generally saw that things went smoothly. No technical book can reach print without the assistance of technical experts, who help ensure that what the author writes resembles reality as closely as possible. This book's reviewers were Karel Baloun, Amy Fong, Howard Lee Harkness, Harold Hauck, Eric H. Herrin II, David King, Rob Kolstad, Matthew Miller, Ian Redfern, and Alexy Zinin. If any errors remain after the text ran their gauntlet, those errors are, of course, my own. Even aside from his help in technical reviewing, I'd like to thank David King for many helpful discussions about Linux networking. Finally, I'd like to thank my agent, Neil Salkind at Studio B, who helped get this book off the ground, with help from Michael Slaughter at Addison-Wesley.

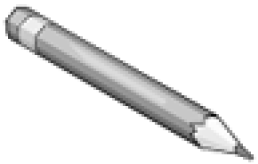
Part I: Low-Level Configuration

Chapter 1. Kernel Network Configuration

"All roads lead to Rome," the saying goes. Something similar is true of Linux networking, except that in this case, Rome is the Linux kernel. Sooner or later, all network traffic passes through the kernel. Given that not all computers or networks are identical, the Linux kernel includes several options you can set to optimize a system for your specific needs. You can set some of these options by passing parameters to the kernel, either during the boot process or after the system has booted, and many of these cases are covered in subsequent chapters of this book. In other cases you must recompile your kernel to activate a needed option or to deactivate one that might degrade your system's performance.

This chapter is devoted to discussing these kernel configuration options. First up is a discussion of kernel configuration procedures. Next is information on network protocol options, such as TCP/IP features, network filters, and support for non-TCP/IP protocol stacks. Next comes a discussion of Linux's drivers for various types of network hardware. The chapter concludes with a brief overview of the process of kernel compilation and use.

NOTE

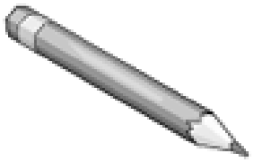


This chapter does not attempt to teach you everything you need to know to compile a kernel; instead, it focuses on the networking options in the kernel. The "Starting Kernel Configuration" and "Compiling and Installing a Kernel" sections include some discussion of more general kernel configuration and use, but if you're new to kernel compilation, you may want to consult the Linux Kernel HOWTO (http://www.linuxdoc.org/HOWTO/Kernel-HOWTO.html) or the kernel compilation chapter of an introductory Linux book.

Starting Kernel Configuration

To configure compile-time kernel options, you must begin with the kernel source code. All major distributions ship with this, but it may or may not be installed by default. Many distributions make changes to the standard kernel (say, to add new drivers that aren't yet standard). You may prefer to start with a standard kernel and add only those patches you need (it's possible you won't need any). Check http://www.kernel.org or a major Linux archive site like ftp://sunsite.unc.edu for the latest kernel source code. (You can also obtain kernel source code from your Linux distribution, but many distributions ship with kernels that have been patched to include non-standard drivers. Using a more standard kernel can be beneficial if you run into problems and need help solving them.)

NOTE



There are two current branches of kernel development, which are distinguished by the second number in the three-part version number. Those with even second numbers (like 2.4.17) are known as *stable* or *release* kernels. Kernels with odd second numbers (like 2.5.2) are *development* kernels. Stable kernels are best for production environments, because they are, as the name implies, quite reliable. Development kernels, on the other hand, are being actively tinkered with—the kernel developers use this line to add new drivers, change interfaces, and so on. Development kernels are therefore best avoided unless you want to contribute to kernel development or if you really require some new driver. (In the latter case, you can often find a *back-port* of the driver to an older stable kernel.)

Kernel source code normally resides in /usr/src/linux, or in a subdirectory of /usr/src that includes the kernel version number, like /usr/src/linux-2.4.17. In the latter case, it's common practice to create a symbolic link called /usr/src/linux and point it to the true Linux source directory. This allows other programs that assume the source is in /usr/src/linux to function correctly, even if you want to keep multiple versions of the kernel source code; you can simply change the symbolic link as required.

Once you've uncompressed the kernel source code into /usr/src/linux, you should change to that directory in a normal command shell. You can then issue a command to configure the kernel options. Possibilities include the following:

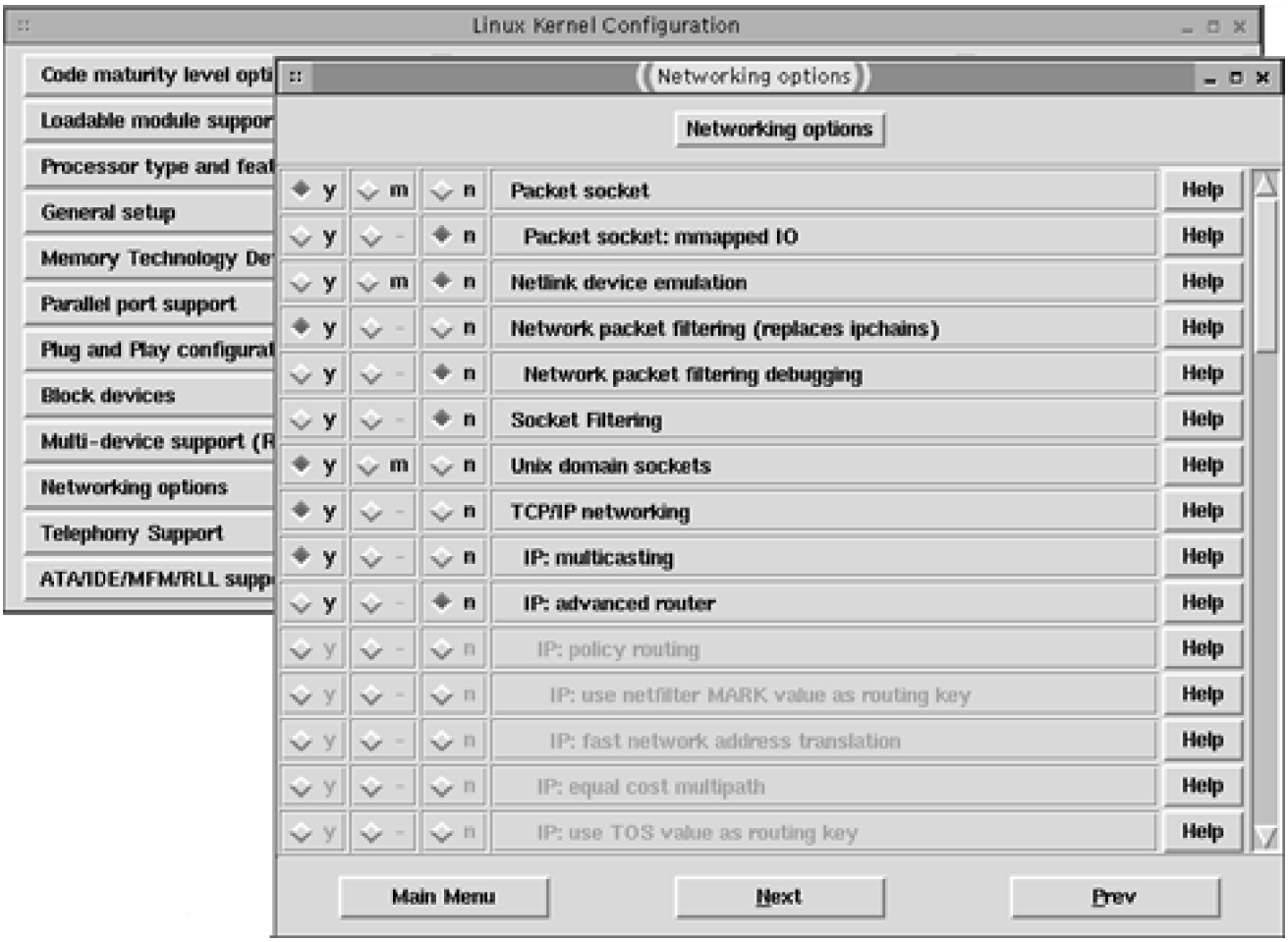
- **make config**— This is the basic configuration tool. It asks you about every kernel option in turn, which can be tedious. If you make a mistake, you must normally go back and redo everything. For this reason, it's seldom used today.
- **makemenuconfig**— This configuration procedure uses text-based menus for configuration

options, which enables you to look through the options and adjust only those that require changes. This is a common method of configuration in text-mode environments.

- **make xconfig**— This method is similar to **make menuconfig**, except that **make xconfig** uses GUI configuration menus. You can click on a topic to see its options, then click your mouse to select how or if you want to compile any option. This is a popular means of kernel configuration when the X Window System (or X for short) is running.

All of these methods present the same options, which are organized into broad categories. (Some categories also include subcategories.) When you select one category with **make menuconfig** or **make xconfig**, a new menu appears showing the options within that category. (Figure 1.1 shows this for **make xconfig**.) Of particular interest for networking are the Networking Options and Network Device Support categories, which are the subject of the next two sections.

Figure 1.1. Linux kernel compilation options are organized into categories and subcategories, each with its own menu.



NOTE

This chapter describes the Linux 2.4.*x* kernel options, and particularly those in the 2.4.17 kernel. Kernel network options have changed in

the past, and are likely to do so again in the future. 2.2.*x* kernels use similar options, but several details differ. A new kernel configuration tool, known as CML2, is under development in the experimental 2.5.*x* kernels. Check http://tuxedo.org/~esr/cml2/ for more information on it.

Most kernel options use a two- or three-way toggle (the Y, M, and N options shown in Figure 1.1). Y and N refer to the option's presence or absence in the kernel file itself, respectively, and M stands for *modular compilation*—compiling the option as a separate file that can be loaded and unloaded. These options are described in more detail in the upcoming section, "Drivers: Modules or Built-In."

Network Protocol Support

The Networking Options kernel menu contains options related to network protocols. You can include or exclude support for entire protocol stacks, and for some (particularly TCP/IP), you can fine-tune the support to optimize the kernel for particular roles, such as router options or packet filtering.

Packet and Socket Options

At a fairly low level, Linux networking operates by allowing programs to send or receive chunks of data (known as *packets*) via data structures known as *sockets.* In most cases, a program opens a socket in a manner that's similar to the way a program opens a file. The program can then send and receive data via that socket. The network protocol stack (discussed shortly in "Alternative Network Stack Options") processes the data in ways that allow it to reach its destination or to be interpreted by the program after having been received from the sender.

In some cases, it's desirable or even necessary to process network data in some other way, or to modify or extend the standard packet and socket operations. Some of these options are important enough that they have their own sections. A few miscellaneous options include the following:

- **Packet Socket**— This option allows applications to bypass much of the normal protocol stack. Most programs don't need this feature, but some network diagnostic tools and other low-level utilities do need it. For instance, tcpdump, which displays low-level TCP/IP packet information, uses this kernel option. Including this option unnecessarily will slightly increase the size of the kernel and might allow an intruder to use low-level network diagnostics like tcpdump that you'd rather the intruder not be able to use. Omitting this feature will prevent *you* from running these utilities, though.

- **Packet Socket: Mmapped IO**— This is a packet socket suboption that, if enabled, can improve the performance of tools that use packet socket connections.

- **Unix Domain Sockets**— Several common and important Linux programs use networking protocols to communicate with each other when they run on a single computer. Examples include syslogd (which handles log files) and X (X programs use network protocols to communicate with the X server, which displays their windows). The Unix Domain Sockets option allows this within-computer communication even on systems that lack conventional network hardware. When computers have conventional hardware, the domain sockets approach is faster than using the more general-purpose TCP sockets. You should include this option on all normal Linux systems; only specialized embedded devices or the like might lack this option.

These options all have default settings that are reasonable for most installations. You might want to disable packet socket support on some systems, though.

Network filters are designed to allow the system to block or modify packets that come into or leave a computer. One of these options (packet filtering) is particularly important for constructing firewalls or performing IP masquerading, as discussed in Chapter 25, Configuring iptables. A firewall can block certain types of undesirable access to a computer or a network that it protects, and IP masquerading lets you share a single IP address among an entire network. Specific kernel network filter options include the following:

- **Socket Filtering**— Normally, the kernel passes all packets that it receives for a given socket on to the program that created the socket. This option allows the program to point the kernel to a small program (known as a *filter*) that will block some of the packets it receives. Few programs require this facility, but the Dynamic Host Configuration Protocol (DHCP) is an important exception—both recent DHCP servers and some DHCP clients require it. You should therefore enable this option if your network uses DHCP.

- **Network Packet Filtering**— This option is the 2.4.*x* kernel's most important type of filter, because it enables certain firewall and IP masquerading techniques. Because these are so important, it's generally a good idea to include this support. When you do so, the Network Packet Filtering Debugging option becomes available, which you can enable if you experience problems. A later submenu, IP: Netfilter Configuration, also becomes available. Subsequent items in this list appear on this submenu.

- **Connection Tracking**— Enabling this option allows the kernel to track network connections in greater detail than is normal. For instance, a router usually passes packets more-or-less blindly between two network interfaces, but when this option is enabled (both in the kernel and by user-level tools), Linux can match up the source and destination IP addresses and ports for future reference. This feature is required for IP masquerading, so it should be enabled on a computer that is to function in this way. It's not necessary for most other systems. If you enable it, the FTP protocol support option becomes available. FTP requires extra housekeeping, so enable this option if you want to use FTP on an IP masqueraded connection.

- **IP Tables Support**— This option includes kernel support routines for the iptables utility, which is used to set up packet filter firewalls and IP masquerading, as discussed in Chapter 25. Activating this option also allows you to select a number of suboptions that fine-tune the features available to you. Many of these options have names of the form *Criterion Type* Match Support, which enables the kernel to match on the specified *Criterion Type.* Of these, Connection State Match Support is particularly important, because it allows the system to perform *stateful packet inspection,* a useful form of firewall operation discussed in Chapter 25. The Packet Filtering, Full NAT, and LOG Target Support options are also very important, as are each of their suboptions. Enable all of these features if you want to use a computer as an IP masquerading router or firewall. You can omit Full NAT for a standalone

workstation or server.

- **ipchains (2.2-Style) Support**— If you have an older firewall script that's based on the ipchains utility used by the 2.2.*x* kernels, you can activate support for this utility as long as you don't compile IP Tables Support directly into the kernel. (The ipchains and iptables tools are mutually incompatible methods of doing largely the same things, but iptables is more advanced.) If you're creating a firewall from scratch, you can safely omit ipchains support.

- **ipfwadm (2.0-Style) Support**— The 2.0.*x* kernels used a firewall tool called ipfwadm. If you have an ipfwadm-based firewall script, you can use it by compiling this feature, which is incompatible with both the iptables and ipchains support. Unless you have such a script and lack the inclination to modify it to use iptables, you can safely omit this option.

Between the 2.0.*x* and 2.4.*x* kernels, Linux's network filtering options have become more sophisticated. The 2.4.*x* kernel includes many optional features, and it's important that you activate all those you'll need for the type of firewall you intend to implement. When in doubt about a specific feature in the IP: Netfilter Configuration menu, I recommend you activate it. This will increase the kernel's size slightly, but it will also provide you with greater flexibility in designing firewall rules.

WARNING

You may think that you don't need to implement firewall rules on a Linux computer, particularly if it resides on a network behind a dedicated firewall. Unfortunately, even many allegedly protected networks have security flaws, so it's best to err on the side of caution. To that end, implementing simple firewalls on individual Linux computers is often a good idea.

TCP/IP Routing Options

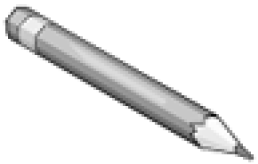
A *router* (also referred to as a *gateway*) is a computer that transfers data between two or more networks. For instance, a department at a large company or university is likely to have a router to link its own subnetwork with the larger network that belongs to the company or university as a whole. The company or university will then have a router to link all of its computers to the Internet. This topic is important enough that Chapter 24, Advanced Router Options, is devoted to the subject. For now, know that the Linux kernel includes several options that can influence its router operation. These are clustered as suboptions of IP: Advanced Router. Chapter 24 discusses the configuration and use of these options in more detail.

IPv6 Support Options

The Internet is built on TCP/IP protocols, and particularly on version 4 of the IP protocols (*IPv4*). Unfortunately, IPv4 is showing its age in many ways. For instance, it supports IP addresses that are four bytes (32 bits) in length, meaning that there is a theoretical maximum of 2^{32}, or 4,294,967,296, addresses. Because of inefficiencies in the way addresses are assigned, the number of Internet addresses is actually much lower than this. Consequently, the Internet is running out of addresses. IPv4 also has security limitations that allow miscreants to seriously disrupt Internet operation. These problems are not severe in 2002, but they're likely to become critical well before decade's end.

For these reasons, an upgrade to IPv4, known as *IPv6,* is under development. Among other things, IPv6 uses a 128-bit IP address for a theoretical maximum of 2^{128}, or 3.4×10^{38} addresses—enough for 2.2×10^{18} addresses per square millimeter of land surface on the Earth. IPv6 also includes better hooks for certain types of security systems than does IPv4. In 2002, few networks allow the use of IPv6, but if yours is one, or if you want to experiment with IPv6 on a private internal network, you can activate experimental Linux IPv6 support via the IPv6 Protocol (Experimental) option in the Networking Options menu. Once you do this, another option or two may become available, including an entire submenu entitled IPv6: Netfilter Configuration. This submenu includes a subset of options similar to those described earlier, in "Network Filter Options," but geared towards IPv6 rather than IPv4.

NOTE



In order to activate IPv6 support, you must select Yes for the Prompt for Development and/or Incomplete Code/Drivers option in the kernel's Code Maturity Level Options menu. This is true of other "experimental" drivers as well. Eventually, IPv6 will become mainstream and nonexperimental. Like other experimental features, you should treat IPv6 support with some caution.

QoS Options

Suppose your Linux system is a router for a busy domain, or is a major server that processes a lot of traffic. In situations like this, it's not uncommon for Linux to find that it has more packets to process than it can send over its network interfaces. Thus, Linux needs some system for scheduling the transmission of outgoing packets. Ordinarily, Linux uses a first-in/first-out (FIFO) strategy, in which each outgoing packet waits in line behind all the others that have already been queued. In some situations, however, you might not want to use this system. You might want to favor certain types of packets, such as those delivered to certain networks or those that involve certain protocols. For instance, you might want to favor packets that carry real-time data, such as Internet telephony protocols. Adjusting packet priorities is the job of the *quality of service (QoS)* options. These options are all available from the QoS and/or Fair Queueing menu off of the Networking Options menu.

In order to implement a QoS system, you must select the QoS and/or Fair Queueing option in the

menu of the same name. This action enables many of the options on this menu. A few others rely upon your selection of one or more other specific options. The most basic features are enabled by the various packet scheduler and queue options, such as CBQ Packet Scheduler and SFQ Queue. These options allow the kernel to schedule packets in more complex ways than the default FIFO. The QoS Support and Packet Classifier API options, as well as their individual suboptions, enable the use of Differentiated Services and the Resource Reservation Protocol (RSVP). These both allow routers to communicate QoS priorities to other routers. If all the routers between two sites implement compatible QoS protocols, the end result can be greatly improved performance for time-critical protocols, at the expense of less time-critical protocols.

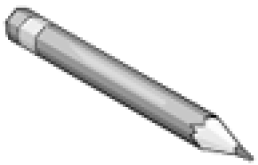
Most nonrouter systems don't need any QoS options. If you're configuring a Linux computer as a router, though—particularly a heavily used router—you may want to activate these options. If you activate one, it may make sense to activate them all, because without all options activated, the tools you use to specify QoS criteria won't be as flexible. For instance, if you omit the U32 Classifier option, you won't be able to prioritize traffic according to the destination address.

In practice, using QoS features requires the use of advanced routing tools, such as ip and tc. Chapter 24 touches upon these tools, but they can be extremely complex. The iproute2 + tc Notes (http://snafu.freedom.org/linux2.2/iproute-notes.html) and Differentiated Services on Linux (http://diffserv.sourceforge.net) Web sites contain additional documentation on these tools.

High-Level Protocol Support

The Linux kernel includes explicit support for several high-level network protocols. Placing this support in the kernel has two principal advantages. First, this code can run more quickly than can an ordinary user-level program. Second, placement in the kernel permits a tighter integration of the features of that protocol with the rest of the system. For instance, kernel-level support for network file-sharing protocols allows Linux to mount remote file exports as if they were local filesystems. The 2.4.*x* kernel includes support for three particularly important high-level protocols: HTTP, NFS, and SMB/CIFS.

NOTE



This list of protocols is not comprehensive. Several others (particularly for network file-sharing protocols) are supported.

HTTP Acceleration

The Hypertext Transfer Protocol (HTTP) is at the core of the World Wide Web. Beginning with the 2.4.*x* kernels, Linux includes what is effectively a simple HTTP server in the kernel. This server is included with the Kernel HTTPd Acceleration option and configured and activated by writing specific

values to pseudofiles in the /proc/sys/net/khttpd directory, as described in Chapter 20, Running Web Servers.

The kernel's HTTP server was created because the work of serving static Web pages (that is, those whose contents are fixed, as opposed to dynamic pages whose contents may be customized for individual users) is essentially just one of copying files from disk to a network address. This operation can be performed much more efficiently in the kernel than in a user-space program. For dynamic content and even some types of static content, the kernel's server falls back on a user-space Web server such as Apache. No special Apache configuration is required; Apache simply doesn't see requests for static Web pages.

NFS Options

Sun developed the Network Filesystem (NFS) as a way to share files among several computers as if those files were local. Linux includes support for NFS, as detailed in Chapter 8, File Sharing via NFS. To mount remote NFS exports, you must include NFS support in the kernel. Most Linux NFS servers also rely on support in the kernel. Both client and server NFS options reside in the Network File Systems submenu off of the File Systems menu, not in the Networking Options menu. Specifically, options you might want to activate include the following:

- **NFS File System Support**— This option enables basic NFS client support (that is, the ability to mount remote NFS exports as if they were local disk partitions). Enable it if you want to mount NFS directories exported by other computers.

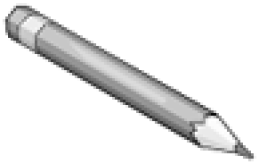
- **Provide NFSv3 Client Support**— NFS has undergone various revisions, the latest of which is version 3 (NFSv3). This support must currently be explicitly enabled, because it's not as reliable as is support for older versions of NFS, as activated by NFS File System Support. The NFSv3 support relies on the basic NFS support.

- **Root File System on NFS**— If you select IP: Kernel Level Autoconfiguration in the Networking Options menu, you can select this option, which lets Linux mount its root filesystem from an NFS export. You'll normally only use this option on workstations that lack hard disks.

- **NFS Server Support**— To have Linux function as an NFS server (that is, to make some or all of its directories available to other computers), you need to run an NFS server. This option provides acceleration features for NFS servers that are written to take advantage of it. This option is not strictly required to run an NFS server, but it's generally a good idea to include it, since most Linux NFS servers are written to expect this support.

- **Provide NFSv3 Server Support**— If you want to run a kernel-aware NFS server for clients that understand NFSv3, activate this option. As with NFSv3 client support, this option relies upon the matching generic NFS support.

NOTE



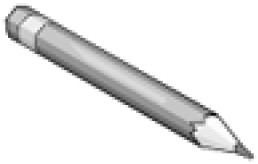
NFS is used mainly by Unix and Linux systems. File sharing between other platforms is usually handled by other tools, one of which is discussed next.

SMB/CIFS Options

NFS isn't the only network file-sharing protocol available. Macintoshes often use AppleTalk, for instance, and Novell's IPX/SPX is a popular protocol stack with associated file-sharing tools. Perhaps the most common file-sharing tool for Linux, aside from NFS, is Samba, which implements the Server Message Block (SMB) protocol, which is also known as the Common Internet Filesystem (CIFS). Chapter 7, File and Printer Sharing via Samba, covers Samba configuration and use.

Samba provides everything needed for Linux to function as an SMB/CIFS server, so there's no kernel configuration required for this function. If you want Linux to be able to mount SMB/CIFS shares, though, you must activate the SMB File System Support option, which is roughly equivalent to NFS File System Support for NFS. Two suboptions (Use a Default NLS and Default Remote NLS Option) let Linux perform filename translations based on National Language Support (NLS) character sets. These options may be important if you use non-Roman alphabets like Cyrillic, or even extensions to the Roman alphabet as used by English, like characters that contain umlauts.

NOTE



It's possible to use Linux as an SMB/CIFS client using the smbclient program, even if you don't activate Linux's SMB/CIFS kernel options. smbclient doesn't actually mount an SMB/CIFS share, though; it gives you access to the share using an FTP-like interface.

Alternative Network Stack Options

Although TCP/IP is the most popular set of network protocols for Linux, and the one upon which the Internet is built, it's not the only choice of network protocol stack. The Networking Options menu includes several others. Most of the options in this menu are actually suboptions of TCP/IP Networking. If you scroll past these, you'll see the alternatives to TCP/IP:

- **Asynchronous Transfer Mode (ATM)**— This is an experimental set of options to support ATM hardware and protocols. ATM is really at least as much of a hardware definition as a network stack, but in the 2.4.*x* kernels, it's enabled in the Networking Options menu, along with other protocol stacks.

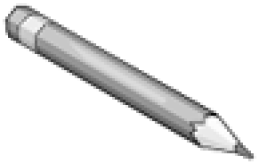
- **The IPX Protocol**— Novell's Internetwork Packet Exchange (IPX) is a protocol stack that's used on many local networks, particularly those running the Netware server OS. To use this stack, you'll need additional software, such as Mars_nwe (documented at http://www.redhat.com/support/docs/tips/Netware/netware.html). The NCP File System Support option in the Network File Systems submenu of the File Systems menu will let you mount Netware volumes, much as the equivalent NFS and SMB/CIFS options let you mount NFS exports or Windows file shares.

- **AppleTalk Protocol Support**— Apple developed the AppleTalk protocol stack to enable file and printer sharing on its Macintosh computers. Linux supports AppleTalk through a combination of the kernel and the Netatalk package (http://netatalk.sourceforge.net/).

- **DECnet Support**— Digital Equipment Corporation (DEC; since bought out by Compaq) developed a network technology known as DECnet for its computers. Linux includes support for DECnet, but you must have a package of programs to use this protocol stack. Check http://linux-decnet.sourceforge.net for more information.

Linux also includes support for a handful of more obscure network protocols, such as Acorn's Econet. On most systems, TCP/IP and possibly one or two other protocols will be quite sufficient. Because of the success of the Internet, vendors who had previously used proprietary protocol stacks have been converting their tools to use TCP/IP. For instance, although Apple has long used AppleTalk, its file-sharing tools now work both over plain AppleTalk and a TCP/IP-based variant.

NOTE



The standard Linux kernel lacks support for one common network stack, NetBEUI. This stack was the default for Windows file sharing via SMB/CIFS in the past, but SMB/CIFS today works equally well over TCP/IP.

Chapter 3, Alternative Network Stacks, covers network stacks and their use in more detail.

Network Hardware Options

The Network Device Support kernel menu contains options related to network hardware. The most important of these options are drivers for specific network cards. The most common types of network cards today are Ethernet devices, but others include traditional local network hardware, long-distance devices, and wireless devices. PC Card devices (for notebook computers) have their own submenu off of the Network Device Support menu. You also select dial-up devices (used to establish connections over telephone modems and some other types of hardware) here.

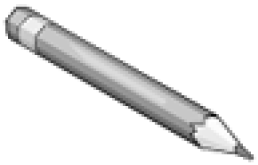
Most of these devices require that you select the Network Device Support option at the top of the Network Device Support menu. If you fail to do this, other options won't be available.

Ethernet Devices

Ethernet is the most common type of local network hardware in 2002, and it seems likely to retain that status for some time. (Wireless technologies, discussed shortly, are becoming popular in some environments, but they lag behind Ethernet and several other wired technologies in terms of speed.) From the point of view of an OS, the problem with Ethernet's popularity is that it's spawned literally hundreds, if not thousands, of specific Ethernet cards.

Fortunately, most Ethernet cards use one of just a few chipsets, so Linux can support the vast majority of Ethernet cards with about 60 drivers. These drivers are split across two submenus: the Ethernet (10 or 100 Mbit) and Ethernet (1000 Mbit) menus. By far the most drivers appear in the first menu, which as the name implies covers 10 and 100Mbps devices. (The most popular type of Ethernet in 2002 is 100Mbps, although 1000Mbps, or *gigabit Ethernet,* is gaining in popularity, and 10 gigabit Ethernet is being developed.)

NOTE



In addition to three common Ethernet speeds, there are several different types of Ethernet cabling: coaxial (used only with some forms of 10Mbps Ethernet), twisted-pair (used by some types of 10Mbps, all types of 100Mbps, and some forms of gigabit Ethernet), and fiber-optic (used by some forms of gigabit Ethernet). Twisted-pair cabling supports distances of up to 100 meters (m) between devices (one of which is normally a central hub or switch), and fiber-optic cabling permits distances of up to 5 kilometers (km) between devices.

The organization of the 10 or 100Mbps driver menu is less than perfect. The menu begins with listings for several popular or once-popular devices from 3Com, SMC, Racal-Interlan, and a few other companies; proceeds with a grouping of Industry Standard Architecture (ISA) bus cards; continues

sample content of Advanced Linux Networking

- read The October Revolution in Prospect and Retrospect: Interventions in Russian and Soviet History (Historical Materialism Book Series, Volume 37)
- **read Misleid (Van In, Book 24)**
- A Necklace of Water (Balefire, Book 4) for free
- **click Aviation Visual Perception: Research, Misperception and Mishaps book**

- http://unpluggedtv.com/lib/Death-Wore-White.pdf
- http://studystrategically.com/freebooks/Corpses-and-Cogitos-and-the-Sympathetic-Self--Exhuming-Sovereignty-and-its-Sympathetic-Subjects.pdf
- http://qolorea.com/library/Pathfinder-Chronicles--Into-the-Darklands.pdf
- http://serazard.com/lib/Red-Eagles--Americas-Secret-MiGs.pdf