SCO® TCP/IP Runtime System for SCO® UNIX® Systems

User's and Administrator's Guide

SCO® TCP/IP Runtime System for SCO® UNIX® Systems

User's and Administrator's Guide

SCO TCP/IP is derived from Interactive Systems Corporation
SYSTEM V STREAMS TCP/IP, a joint development of ISC and Convergent Technologies.

Date: 15 May 1992
Document version: 1.2.0A

Chapter 8

Administering serial line communications 55

Chapter 9

Configuring the BIND name server 73

Chapter 14

TCP/IP sendmail administration 159

Preface

SCO® TCP/IP is a set of protocols and programs used to interconnect computer networks and to route traffic among different types of computers. It provides the following key services:

- data transfer protocols that applications such as mail or SCO NFS can use to move data from machine to machine
- programs that allow the user to log in remotely to other computers on the network, print remotely, transfer files, and perform other network-based tasks
- protocols and programs that provide for network management and troubleshooting, such as the Simple Network Management Protocol (SNMP) and the Berkeley Internet Name Domain (BIND) Server

About this guide

The *SCO TCP/IP User's and Administrator's Guide* provides functional descriptions of TCP/IP components and steps for TCP/IP configuration. Chapters 2 through 6 are intended for end users; chapters 7 through 15 are intended for system administrators and others with an interest in the administration and configuration of TCP/IP.

Chapter 1, "Using and Administering TCP/IP," provides conceptual information about networking and how TCP/IP works, such as descriptions of the Internet Protocol and a discussion of installation concepts. Chapter 1 also gives you network planning ideas, and we strongly suggest that you read this chapter before installing TCP/IP.

Chapter 2, "Logging into a remote machine," explains how to use the **rlogin** and **telnet** commands to access another machine on the network.

Chapter 3, "Transferring files between machines," shows how you can use **ftp** and **rcp** to move files from one networked machine to another.

Chapter 4, "Running commands remotely with rcmd," tells you how to run a command on another machine from your machine.

Chapter 5, "Sending mail across the network," provides a brief introduction to the **mail** command.

Chapter 6, "Other useful commands," lists several other user-level TCP/IP commands you may find useful.

Chapter 7, "Administering TCP/IP," describes many of the basic TCP/IP administration tasks, such as establishing user equivalence and adding pseudo-ttys.

Chapter 8, "Administering serial line communications," describes serial line communications over TCP/IP, including the SLIP and PPP protocols.

Chapter 9, "Configuring the BIND name server," shows how to configure the Berkeley Internet Name Domain Server, a distributed host name and address lookup system.

Chapter 10, "Gateways and routing," explains how to set up your system as a gateway computer through use of **gated** and **routed**.

Chapter 11, "Configuring and using SNMP," describes the Simple Network Management Protocol, a set of programs by which you can monitor and troubleshoot your network.

Chapter 12, "Remote line printing," describes how to enable remote printing over TCP/IP.

Chapter 13, "Synchronizing clocks," explains the two time protocols you can configure for use with your network.

Chapter 14, "Configuring sendmail," explains how to configure **sendmail**, one of the mail routers supported by TCP/IP.

Chapter 15, "Helpful Hints," provides answers to several common troubleshooting questions.

The "Bibliography" describes related reading that provides further information about TCP/IP.

Conventions used in this guide

This guide uses the following notational conventions:

bold	represents commands, command options, parameters in files, data structures, and daemons
BOLD CAPS	represents parameters contained in files
italics	represents files and directories
bold italics	represent variables that you supply; for example, in the command argument **path:*pathname***, the variable ***pathname*** is replaced with an actual pathname when you type the command
⟨ ⟩	represents special keys that you press; for example, ⟨Ctrl⟩**x** means to hold down the Control key and press the **x** key simultaneously, then release them
`Courier`	represents system responses, excerpts from files, and programming examples

Reference pages

Reference pages, also called manual pages or man pages, are descriptive pages for commands, daemons, and files, and other items related to a given product. Reference pages can be viewed online using the **man** command. For example, to get information about the **tar** command, you enter **man tar** at the UNIX® system prompt.

Commands that have reference pages have one or more letters associated with them, such as **tar**(C). The letters in parentheses tell you which reference page section to look in to find information on that command. The letters also tell you which product the command belongs to. For example, commands with the (ADM) suffix are UNIX system administration commands. Commands with the (ADMN) suffix are TCP/IP administration commands. The following letters are relevant to TCP/IP:

C	UNIX system user-level commands
ADM	UNIX system administration commands
ADMN	TCP/IP network administration commands
ADMP	TCP/IP network protocols and drivers
SFF	TCP/IP network file formats
TC	TCP/IP user-level commands

For information on manual pages that have the (ADMN), (ADMP), (SFF), or (TC) suffixes, for example **rlogind**(ADMN), refer to the *SCO TCP/IP Command's Reference.* For all other commands, check your *SCO UNIX System V/386 User's Reference* for a list of manual page sections and their abbreviations.

Related reading

Refer to the other manuals in this set for more information on the various aspects of TCP/IP:

- The *SCO TCP/IP Release and Installation Notes* describe how to install TCP/IP and provide the latest information on the product.
- The *SCO TCP/IP Command Reference* describes all administrative and user-level commands, daemons, and files associated with TCP/IP.

Chapter 1

Networking and TCP/IP overview

This chapter describes networking in general and TCP/IP in particular. After you read this chapter, you will have a better understanding of the components that make up the TCP/IP package, and you can pick and choose which components you want to configure. We strongly recommend that you read this chapter before installing any networking software.

Networking, simply put, is connecting your computers together so they can share information. Effective networking increases productivity by using computer resources, such as files, printers, and memory, more efficiently. A network puts the power of all of your system's hardware and software at your fingertips.

Although there are many different types of networks, they fall into two general categories: local area networks (LANs) and wide area networks (WANs).

A LAN connects computers that are in the same office or in adjacent buildings. All the computers on a LAN are connected to a single cable. A computer on a LAN can communicate directly to any other computer on that LAN. One LAN may also be connected to another LAN via a gateway computer.

A WAN connects computers that can be as close as several hundred feet to as far as across the globe. These connections are made using phone lines and sometimes satellite connections, if the distance is great enough. Sometimes a computer must go through one or more computers, or gateways, to reach the one with which it wants to communicate.

Most networks are a combination of local and wide area networks. Figure 1-1 displays a portion of a typical local area network. It includes several *client* computers, a *server* computer, a printer that is accessible to any machine on the network, an Ethernet ™ cable connecting the machines, and a computer running SCO Open Desktop®, which may be a client, a server, or both.

Servers, often the most powerful computers on the network, store data that they make available to *clients*, other machines on the network that have access to the servers' resources. You can have one or more servers on a network, and a machine can be both a client and server. For example, one machine can serve personnel information while another serves sales data. Each machine is, therefore, a server, but each may also be a client to the other machine's data.

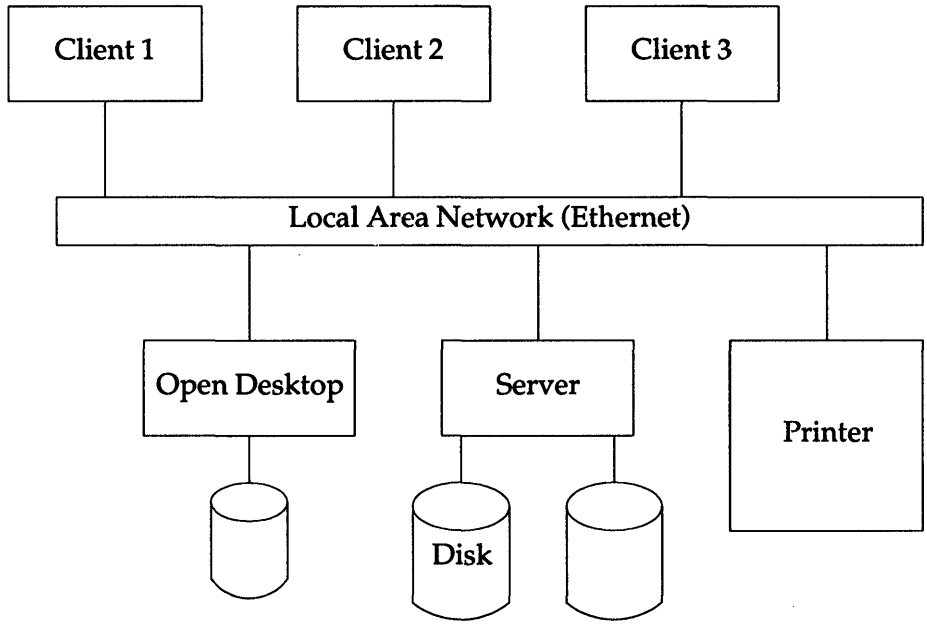


Figure 1-1 Sample network

How the network works

A network, in the physical sense, consists of cables or phone lines. These lines connect the computers, and networking cards provide the means to talk across them. However, a network is not useful unless it has programs on each computer that let humans access the various computers on the network.

Computers on a network have agreed ways of communicating called *protocols*. Protocols dictate which signals computers use across cables, how they tell one another that they have received information, and how they exchange information.

Protocols are more accurately termed protocol suites or protocol families. This subtle shift in terminology reflects the fact that the communications functions are complex and are usually divided into independent layers, also called levels. The protocol associated with each layer communicates with only the layers immediately above and below it, and assumes the support of underlying layers.

In protocol suites, lower layers are closer to the hardware and higher layers are closer to the user. The number of layers and tasks that the layers perform depends on who defines them. TCP/IP has four software layers built on an underlying hardware layer. Its model is shown in table 1-1:

Table 1-1 TCP/IP Model

Layer	Name	Task
4	Application	Accesses the transport layer, and sends and receives data
3	Transport	Provides communication protocols between application programs and the network layer
2	Network	Takes care of communication between software and hardware
1	Physical	Accepts and transmits data over the physical network

Designing or adding to your network

Your machine may be a part of an entirely new network, or it may become a machine on a network that already exists. In either case, you need to make several decisions about your machine:

- With what other computers does it need to communicate?
- Will it serve as a client, a server, or both?
- Who will use this machine, and what sort of access do they need?

Common networking administration tasks

After you decide how your machine fits into the network, you need to install and configure the appropriate TCP/IP packages as described in the *SCO TCP/IP Release and Installation Notes.* You also need to update the networking files on other machines so that they know of the new machine's existence. This configuration ensures, among other things, that:

- all machines on the network know each other's names and addresses
- individual users will have access to files and accounts on various machines
- electronic mail is routed correctly
- the network runs at peak efficiency

Common tasks that you will perform to ensure these goals include:

- installing and maintaining networking hardware and software
- assigning names and addresses to each computer and device on the network
- assigning names and identification numbers (IDs) to network users and groups
- performing the commands required to share, remove, and restrict resources
- updating all appropriate networking files on your network's machines

Introducing TCP/IP

TCP/IP is the set of protocols and programs used to interconnect computer networks and to route traffic among different types of computers. "TCP" stands for Transmission Control Protocol, and "IP" stands for Internet Protocol. These protocols describe allowable data formats, error handling, message passing, and communication standards. Computer systems that use TCP/IP speak a common language, despite any differences in the hardware and software of the various machines.

Many large networks conform to these protocols, including the DARPA Internet (Defense Advanced Research Projects Agency Internet). A variety of universities, government agencies, and computer firms are connected to an internetwork that follows the TCP/IP protocols. Thousands of machines are connected to this Internet, or network of networks. Any machine on the internet can communicate with any other. Machines on the internet are referred to as *hosts* or *nodes*, and are defined by their internet (or IP) address. Defining an internet address is described later in this section.

TCP/IP provides the basis for many useful services, including electronic mail, file transfer, and remote login. Electronic mail is designed to transfer short text files. The file transfer application programs transfer very large files containing programs or data. They also provide security checks controlling file transfer. Remote login allows users on one computer to log in at a remote machine and carry on an interactive session. The TCP/IP programs that facilitate these services are described in detail later in this guide.

The Internet Protocol (IP)

The Internet Protocol, IP, defines a data delivery system wherein the sending and receiving machines are not necessarily directly connected. IP splits data into packets of a given size, which are then forwarded to the receiving machine via the network. These individual packets of data (often called *datagrams*) are routed through different machines on the internet to the destination network and receiving machine. A particular set of data, such as a file, can be broken up into several datagrams that are sent separately. When you use IP to forward datagrams, individual datagrams may or may not arrive, and they probably will not arrive in the order in which they were sent. TCP adds the reliability that IP lacks.

A datagram consists of header information and a data segment. The header information routes and processes the datagram. Datagrams can be further fragmented into smaller pieces, depending on the physical requirements of the networks they cross. For example, when a gateway sends a datagram to a network that cannot accommodate the datagram as a single packet, the datagram must be split into pieces that are small enough for transmission. The datagram fragment headers contain the information necessary to reassemble the fragments into the complete datagram. Fragments do not necessarily arrive in order; the software module implementing the IP protocol on the destination machine must reassemble the fragments into the original datagram. If any fragments are lost, the entire datagram is discarded.

The Transmission Control Protocol (TCP)

The Transmission Control Protocol, TCP, works with IP to provide reliable delivery. It provides a means to ensure that the various datagrams making up a message are reassembled in the correct order at their final destination and that any missing datagrams are resent until they are correctly received.

The primary purpose of TCP is to avoid the loss, damage, duplication, delay, or misordering of packets that can occur under IP. Also, security provisions such as limiting user access to certain machines can be implemented through TCP.

TCP provides reliability using checksums (error detection codes) on the data, sequence numbers in the TCP header, positive acknowledgment of data received, and retransmission of unacknowledged data.

Other TCP/IP protocols

The protocols listed in Table 1-2 are provided as part of TCP/IP:

Table 1-2 Additional TCP/IP protocols

Protocol	Purpose
Address Resolution Protocol (ARP)	ARP translates between DARPA Internet and Ethernet addresses.
Internet Control Message Protocol (ICMP)	ICMP is an error-message and control protocol used by TCP/IP.
Point-to-Point Protocol (PPP)	PPP provides both synchronous and asynchronous network connections.
Reverse Address Resolution Protocol (RARP)	RARP translates between Ethernet and DARPA Internet addresses.
Serial Line Internet Protocol (SLIP)	SLIP enables IP over serial lines.
Simple Mail Transport Protocol (SMTP)	SMTP is used by MMDF to send mail via TCP/IP.
Simple Network Management Protocol (SNMP)	SNMP is the protocol used to perform distributed network management functions via TCP/IP.
User Datagram Protocol (UDP)	UDP provides data transfer without many of the reliable delivery capabilities of TCP. UDP is less CPU-intensive than TCP, and is useful when guaranteed data delivery is not of paramount importance.

These protocols are described in further detail later in this guide.

TCP/IP end-user commands

Several TCP/IP commands, described in detail in chapters 2 through 6 of this guide, provide end users with networking capabilities. Table 1-3 is a partial list of these commands:

Table 1-3 TCP/IP commands

Command	Purpose
ftp	file transfer between machines running TCP/IP; these machines may or may not be running the same operating system
rcmd	remote command execution on another UNIX machine
rcp	file copying between two UNIX machines
rlogin	remote login on another UNIX machine
ruptime	status display of local network machines
rwho	displays list of users logged on to local network machines
telnet	remote login on a machine running TCP/IP; these machines may or may not be running the same operating system

Configuring TCP/IP

This section provides information on software and hardware prompts you need to answer as you configure TCP/IP. We strongly recommend that you read and understand this section before you attempt to install your software. Installation prompts include:

- system's host name and domain name
- Internet address(es) for each driver, adapter, or serial line
- broadcast address
- netmask
- gateway status
- hardware information, including interrupt vectors, base memory addresses, RAM buffer sizes and base addresses, ROM base addresses, DMA channels, and slot numbers

System name

Your system name, or host name, should be unique on your network. It can consist of lowercase letters and numbers, must begin with a letter, and should be no longer than eight characters. **mail** and other programs use the system name to identify the correct data destination. Here are some sample valid machine names: *scosysv, tcpdev, account1*.

Driver type

The driver is the software that allows your networking cards or hardware to interact with TCP/IP. Each card, adapter, slip or ppp line that you use must be uniquely associated with a particular device driver. You can install up to four Ethernet cards of one type, up to two Token Ring adapters, and up to eight serial line interfaces (four SLIP and four PPP), but you can only configure one driver at a time. When you are prompted for the driver type, choose the type you want to configure.

Interrupt vector

Each driver on your system, including those for network cards and SLIP lines, must have its own interrupt vector, or IRQ. This vector must not be used by any other device on the system. Refer to your networking hardware documentation to determine what vectors the hardware supports. In addition, the **hwconfig**(ADM) and **vectorsinuse**(ADM) programs list the hardware already installed on your system and what vectors are already in use, respectively.

Your networking hardware might be pre-configured to use a particular vector. If you want to change this vector setting, you might also need to change the physical jumper settings on the board or run a setup program provided with the board.

NOTE A number of networking cards are pre-configured to use interrupt vector 3. Your operating system has reserved IRQ3 for the sio (serial input-output) device. You can either disable this device during your **netconfig** session, or choose another vector.

I/O base address

Each hardware driver on your system that performs I/O (input/output) needs a unique memory base address so that the system can locate it. This memory address is a three- or four-digit hexadecimal number, must match the settings on the board, and must not conflict with any other hardware on your system. Valid base addresses are displayed when you configure your card.

Thick/thin cable

Some networking cards use thick, rather than thin, networking cable.

- Thin cable provides a direct connection to the network without the use of a transceiver. Most installations use thin cable.
- Thick cable connects your networking card to a transceiver, which in turn connects to the Ethernet cable.

RAM buffer size and base address

Several networking cards require a designated space in RAM to do buffering; you need to specify this address (as a five-digit hexadecimal number) and, if necessary, configure the buffer size.

NOTE The **wdnsetup** command is used to change these values for some Western Digital cards. For more information on this command, see the **wdnsetup**(ADM) manual page.

ROM base address

Several Token Ring cards need a designated space in ROM to store information; see your Token Ring card documentation for more information on available addresses.

Token Ring routing

Token Ring allows you to establish connections from your machine to others on the local ring, or to those on another ring using a bridge. To access those machines on another ring, you must enable Token Ring routing when you configure your Token Ring adapters.

Domain name

The MMDF mail router uses the domain name to route messages, such as mail, from machine to machine. The domain name allows your network to fit into a hierarchical network structure composed of commercial organizations (.COM), educational institutions (.EDU), the government (.GOV), the military (.MIL) or miscellaneous organizations (.ORG). Sample domain names are ***sco.COM*** (the domain name used by SCO) and ***berkeley.EDU*** (the domain name used by the University of California at Berkeley).

Base your domain name choice on the following:

- If other machines on your network already use a domain name, use the same name for the machine you are installing.
- If you are creating a new domain and want to use BIND to connect to the outside world, you need to register the name with the appropriate network (DARPA Internet, CSNET, or BITNET). To register a domain name, write to:

  DDN Network Information Center
  Suite 200
  14200 Park Meadow Drive
  Chantilly, VA 22021

- If you are creating a new domain and might or might not eventually connect to an outside network, use the name ***name*.UUCP**, where name is the name of your company or organization.
- If you will never attach to a network outside your company, choose ***company*.COM**.

IP address

The IP address identifies and differentiates your machine from all others on the network. It consists of a 32-bit binary number that is usually displayed as four octets expressed in decimal and separated by periods. You must have a unique IP address for each machine on your network. In addition, if your machine serves as a *router* to another network, it contains two or more network cards and belongs to two or more networks. In this case, you must assign each card a unique IP address on the appropriate network.

NOTE The IP address differs from an Ethernet address in that it is configurable. An Ethernet address is a 6-byte address that is unique to each physical Ethernet card. This non-configurable address is assigned by the card manufacturer.

The IP address consists of two parts: a network address that identifies the network and a host address that identifies the particular host, or node. Table 1-4 shows an IP address in binary form, as binary octets, as decimal octets, and as it appears in standard notation.

Table 1-4 IP address derivation

binary (32-bit)	10000100100011110000001000000010			
binary (octets)	10000100	10001111	00000010	00000010
decimal octets	132	147	2	2

IP address (in standard notation) = 132.147.2.2

Several classes of TCP/IP networks are available, each based on the number of hosts a network needs. Network classes supported by SCO are Class A, B, and C. Use the smallest network class that can accommodate all of your network's hosts. Most TCP/IP installations use Class C, but some larger installations might need to use Class B.

Table 1-5 lists valid network addresses for each class:

Table 1-5 Internet address classes

Class	Available Hosts per Network	Valid Address Ranges
A	16777216	1.0.0.1 through 126.255.255.254
B	65534	128.0.0.1 through 191.255.255.254
C	254	192.0.0.1 through 222.255.255.254
Reserved		224.0.0.0 through 255.255.255.254

If you are connecting your machine to a pre-existing network, the network address (for Class A, the first octet; for Class B, the first two octets; and for Class C; the first three octets) is the same as those of other machines on the network. In this case, you need only concern yourself with creating a unique host address.

If you are creating an entirely new network and you want to connect to the DARPA Internet, you need to contact the Network Information Center to have a network address assigned. The full address is shown earlier in the section "Domain name". If you do not want to connect to an outside network, you

can choose any network address as long as it conforms to the syntax shown previously. In either case, once you determine the network address, you can then create the unique host address.

When you determine the IP address, keep in mind the following:

- Each logical network must have its own network address.
- All hosts in a network must have the same network address.
- All hosts in a network must have unique host addresses.
- Do not use the following network addresses: 0 or 127 (Class A), 191.255 (Class B), 223.255.255 (Class C), or any of the addresses shown in the Reserved class of Table 1-5.

Broadcast address parameters

All datagrams sent by TCP/IP move through all machines in the network path. However, each host adapter ignores any packet that does not include that particular computer's IP address in the datagram header. Occasionally, you might want to send a message to all machines on a particular network. To do so, select a *broadcast address* for your machine. A broadcast address is one in which the host portion of the IP address consists either of all 0's or all 255's. The configuration procedure prompts you to choose between the following address schemes:

Table 1-6 Broadcast address schemes

Scheme	Example	Purpose
all zeroes (decimal 0)	132.147.0.0	provides compatibility with 4.2BSD systems
all ones (decimal 255)	132.147.255.255	UNIX Operating System Standard (RFC-919)

The addresses shown in the previous table are for a class B network, and are shown as examples only. Your values will be different. If you are on a network that does not contain any machines running 4.2BSD UNIX or earlier BSD versions, choose all ones. If such machines exist on your network, choose all zeroes.

netmask setting

The netmask strips the network ID from the IP address, leaving only the host ID. Each netmask consists of binary ones (decimal 255) to mask the network ID and binary zeroes (decimal 0) to retain the host ID of the IP address. For example, the default netmask setting for a Class B address is 255.255.0.0.

NOTE Always use the default netmask that the installation program prompts you for unless you are creating a subnet, a logical division of a physical network. If you create a subnet, also mask the portion of the address that indicates the subnet. For example, the netmask for a machine on a Class B subnet is 255.255.255.0. For more information on creating subnets, see the chapter "Network administration" later in this guide.

Gateway status

A machine that has interfaces (cards or serial lines) to more than one network may operate as a gateway between networks, by forwarding and redirecting packets from one network to another.

When you configure a second card under TCP/IP, you are prompted to turn this gateway behavior on or leave your machine in the default, non-gateway behavior. If you do not make your machine into a gateway, it will continue to receive packets on each network at the specified IP addresses, but will not forward packets between networks.

TCP/IP over a SLIP or PPP serial line

The following prompts are relevant only to serial line drivers.

tty line

This line indicates what *tty* the SLIP line connects to.

- If you are connecting to COM1:, interrupt vector 4, enter **tty1A**.
- If you are connecting to COM2:, interrupt vector 3, enter **tty2A**.
- If you are connecting to a smart serial card, use the appropriate tty naming convention.

Source IP address

The IP address for this host (this end of the serial line). For more information on determining IP addresses, see "IP address" earlier in this chapter.

Destination IP address

The IP address for the remote host (the opposite end of the line).

Baud rate

The baud rate at which data is transmitted. The default is 9600.

SLIP netmask

A netmask for this SLIP line. For more information on netmasks, see the section "netmask setting" earlier in this chapter.

Maintaining TCP/IP

After you install TCP/IP, you may never need to alter the TCP/IP configuration again. However, there are some common tasks that occur if you want to customize or add to your network. These are described briefly here and in detail later in this guide.

Adding hosts

The */etc/hosts* file is a list of hosts on the network. Network library routines and server programs use this file to translate between host names and Internet addresses when the BIND (Berkeley Internet Name Domain) name server is not being used.

To add a machine to the network, you must add an entry to all of the */etc/hosts* files on the local network. Refer to the *hosts*(SFF) manual page for a description of the file format.

Configuring the name domain server

The Berkeley Internet Name Domain Server (BIND) provides a distributed lookup system for host names and addresses. Enabling BIND overrides the default network information file, */etc/hosts*. For more information, see the chapter "Configuring the BIND name server" later in this guide.

Setting up routing tables

Routing tables provide the information needed to route packets to their destinations properly. For descriptions of several possible approaches to maintaining routing information, see the chapter "Gateways" later in this guide. In addition, the chapter "Network administration" contains a section on obtaining information about the system routing tables.

Establishing user equivalence

You can control who has access to a machine through the network by establishing user equivalence within the */etc/hosts.equiv* and *.rhosts* files. The **rlogin, rcp,** and **rcmd** commands use these files to verify access privileges. For information on how to use these files, see the section "Network databases" in the chapter "Network administration" later in this guide. You can also refer to the *hosts.equiv*(SFF) manual page for a description of the file format. A note in the "Helpful Hints" chapter also discusses user equivalence.

Setting up anonymous ftp

You can set up a public **ftp** account on your system that allows remote users to transfer files anonymously from restricted, public directories on your system. For information on the */etc/ftpusers* file, and a description of how to set up the public **ftp** account, refer to the section "Network databases" in the chapter "Network administration" later in this guide.

Altering installation parameters

You can change many of the settings that you set during TCP/IP installation by altering the appropriate system files (such as */etc/hosts* and device driver files) with a text editor or with an appropriate utility, such as **netconfig, route,** or **mkdev.** The use of such files and utilities, which are documented in the chapter "Network administration," is always preferable to reinstalling the software.

Tuning kernel parameters

You may need to tune kernel parameters by increasing or decreasing STREAMS buffers and other parameters used by TCP/IP. Several utilities, including **netstat, configure,** and **netconfig,** help you fine-tune your system to enhance networking performance. These utilities are described in the chapter "Network administration."

Monitoring TCP/IP status

You can use the **netstat** command to display Internet connections, current Internet activity, routing tables, and error messages, among other useful information. In addition, you can use the Simple Network Management Protocol (SNMP) commands and utilities to further monitor and troubleshoot your network. For more information, see the chapter "Configuring and using SNMP later in this guide.

Enabling remote printing

You can enable the remote printing daemon, **lpd**, to allow print jobs to be sent over the network to remote printers, or to make a printer attached to your computer available to the network. For information on **lpd** and its associated files, see the chapter "Remote line printing" later in this guide.

Chapter 2

Logging in to a remote machine

When you log in to a remote machine over the network, your terminal on the local machine acts as if it were attached to the remote machine. No physical connection is made—software simulates a physical line between your terminal and the remote machine.

Two commands, **rlogin**(TC) and **telnet**(TC), allow you to log in to a remote machine. **rlogin** is very convenient because, when your system is set up properly, you do not have to enter your user name and password to log in to a remote machine; however, **rlogin** only works when you are logging in to a machine that is running a UNIX operating system. **telnet** is not quite as easy to use, but it does not require any setup files and allows you to connect to machines running a variety of operating systems.

You can usually log in to a remote machine successfully using **telnet**, but you will probably find it most convenient to set up your system so that you can use **rlogin** when working with another UNIX system.

Once you invoke **telnet** or **rlogin**, these commands pass to the remote machine all the data that you input, and they display all output from that machine on your screen. When logged in remotely, you can use any command at the command line that you would use when logged in directly, including screen-oriented programs like **vi**(C). (You cannot use icons or perform other graphics-oriented tasks on the remote machine.)

The rlogin command

To log in to another machine running a UNIX operating system, use the **rlogin** command with the name of the remote machine:

rlogin warwick

If system equivalence exists between your local machine and the remote machine (see the section "Establishing user equivalence" in Chapter 7 of this guide) and you have an account on the remote machine, you are automatically logged in with the same user name that you are working with on the local machine. If system equivalence does not exist, you are prompted for a password on the remote machine.

If you want the convenience of automatic login, you can ask the system administrator on the remote machine to establish system equivalence, or you can set up your own user equivalence by creating a *.rhosts* file in your home directory on the remote machine (see the section "Establishing user equivalence" in Chapter 7 of this guide, or the **rhosts**(SFF) manual page).

If your system is configured to allow it, you can log in to another machine simply by entering the name of the remote machine on the command line, without the **rlogin** command. For this to work, your system administrator must create a link in the */usr/hosts* directory for each remote machine, and you must have this directory in your search path.

When you are finished with your work, log out from the remote machine to end the remote terminal session and return to the machine from which you started. **rlogin** tells you that the remote connection has been closed. If, for some reason, you cannot end a remote session normally, type the **rlogin** escape sequence of a tilde followed by a period "~." on a line by itself. This action aborts the remote session and returns you to the local machine.

rlogin command-line options

Some options you can specify when invoking **rlogin** are:

-ec changes the escape character from tilde to the character you specify

-l (lowercase "L") specifies the user name under which you want to log in on the remote machine

-8 allows an 8-bit input data path at all times

Any option must follow the name of the remote machine on the command line. These options are described in more detail in the **rlogin**(TC) manual page.

NOTE After you run **rlogin** with the **-8** option, you still need to specify 8-bit stty settings for the **rlogind** daemon on the remote machine. Therefore, after you log into the remote machine, execute the following command:

stty -istrip

You can run this command from the command line or from a startup file on the remote machine (*.profile* for accounts using the Bourne or Korn shell and *.login* for accounts using the C shell).

Using a tilde in the text

When you are logged in to a remote machine, you cannot normally type a tilde at the beginning of a line because the tilde is the default escape character. If you need to type a line that begins with a tilde, you must type two tildes "~~".

If you change the escape character with the **-e** option, you must type the new escape character twice when you want it to appear at the beginning of a line.

The telnet program

telnet allows you to log in to a remote machine as **rlogin** does, but it is not as convenient to use because it is designed to work with any operating system, not only with a UNIX system. When using **telnet**, you always have to enter a user name and password.

To log in to another machine, use the **telnet** command with the name of the remote machine:

telnet warwick

telnet prompts for a user name and password on the remote machine. When you see the prompt from the remote machine, you can enter commands.

When you log out from the remote machine, you end the remote terminal session and return to the machine from which you started. You can interrupt a remote session at any time by entering the **telnet** escape character ⟨Ctrl⟩-] on a line by itself.

telnet provides a command mode from which you can control **telnet** operations; you can set options that define how your machine communicates with another machine when you are logged in remotely. From command mode, you can also connect and disconnect from a remote machine with the **open** and **close** commands.

To enter **telnet** command mode, give the **telnet** command without a machine name. The **telnet** command prompt looks like this:

```
telnet>
```

At this prompt, you can enter any **telnet** command (enter **?** or see the **telnet**(TC) manual page for a list of commands with descriptions). At any time, the **status** command shows whether or not you are connected to a remote machine, the current option settings (if you are connected to another machine), and the current escape character. The **quit** command ends the remote session and exits from **telnet**. You can abbreviate a command as long as you enter enough characters to distinguish it from other **telnet** commands.

You can also enter command mode by entering the **telnet** escape character, ⟨Ctrl⟩-**]**, while already logged in to a remote machine. The escape character temporarily interrupts the remote login session and places you in command mode so you can execute **telnet** commands. With most **telnet** commands, you automatically exit command mode when the command finishes. With some (such as **?**), you need to press ⟨Return⟩ when the command finishes.

telnet command-line options

Some options you can specify when invoking **telnet** are:

-ec changes the escape character from Ctrl-] to the character you specify

-l (lowercase "L") specifies the user name under which you want to log in on the remote machine

-8 allows an 8-bit input data path at all times

Any option must follow the name of the remote machine on the command line. These options are described in more detail in the **telnet**(TC) manual page.

Chapter 3

Transferring files between machines

Two commands, **rcp** (remote copy) and **ftp** (file-transfer program), allow you to transfer files between machines on the network. **rcp** is very convenient because you do not have to enter your user name and password for the remote machine, and it allows you to copy an entire directory; however, **rcp** can only transfer files with a machine that is running a UNIX operating system, and you must have user or system equivalence with the remote machine. **ftp** is not quite as easy to use, but it allows you to transfer files with machines running a variety of operating systems, and it does not require user equivalence. If you often work with another UNIX system, you will probably find it most convenient to set up your system so that you can use **rcp**.

The rcp command

To use the **rcp** command, the machine with which you want to transfer files must be running a UNIX operating system, and you must have user or system equivalence with the remote machine (see the section "Establishing user equivalence" in Chapter 7 of this guide).

The syntax of the **rcp** command is much like that of the UNIX **cp** command, where you give the name of the file to be copied and the location to which it should be copied. The **rcp** syntax is different from the **cp** syntax in that you can precede either the source path or the destination path with a machine name to specify files on a remote machine. You must separate the machine name from the filename with a colon. The square brackets in this **rcp** syntax description show that the machine name is optional for both the source and destination files:

rcp [*machine*:]*directory.../filename* [*machine*:]*directory.../filename*

You can use **rcp** to copy from a local file to a remote file, or vice versa. For example, to copy the file *proposal* from the directory */u/proj3/design* on the machine *warwick* to the current directory on your local machine, enter:

rcp warwick:/u/proj3/design/proposal proposal

As another example, you can copy your weekly report from your own **status** directory on the local machine to the group status directory on the machine *warwick* with:

rcp /u/perry/status/engr.09.29 warwick:/u/staff/status

To copy a directory, you must use the **-r** option. For example, to copy the */u/proj3* directory from *warwick* with all its subdirectories and files to a subdirectory named *proj3* in your current directory on the local machine, enter:

rcp -r warwick:/u/proj3 proj3

With the **-r** option, the destination must be a directory.

Copying files of other users

You can use the **rcp** command only to access files and directories to which you would ordinarily have access according to UNIX file permissions. **rcp** verifies file access permissions with the user name under which you are logged in on the local machine. If you have user equivalence with another user on the remote machine, you can access that user's files by specifying the user name in the **rcp** command line. For example, if you have user equivalence with *rsimpson* on *warwick*, you can copy a file from that user's directory with:

rcp rsimpson@warwick:personal/letter letter.rsimpson

Because the remote path for the file *letter* is not an absolute path, **rcp** assumes the path is relative to the specified user's home directory.

Copying between remote machines

With the **rcp** syntax, if you specify a machine name for both the source and destination files, you can copy a file between two remote machines without first moving the file to your local machine. From your local machine *blue*, you can use the following command to copy the file *notes* from your home directory on *warwick* to your home directory on *ivy*.

rcp warwick:notes ivy:notes

You must have user equivalence with your accounts on both remote machines.

See the **rcp**(TC) manual page for alternate syntax and other details about this command.

The ftp command

ftp transfers files between machines as **rcp** does, but is not as simple to use because it is designed to work with any operating system, not only with a UNIX system. **ftp** has certain limitations compared with **rcp**, but it also provides certain features that are not available with **rcp**. For example, **ftp** provides the ability to copy both ASCII and binary files with a different operating system and allows certain file-transfer privileges for a user who does not have an account on a machine.

Invoking ftp

ftp is an interactive program with its own set of commands for accessing network files. To invoke **ftp**, enter the **ftp** command without any arguments. You see the **ftp** prompt:

```
ftp>
```

At the **ftp** prompt, you can give any **ftp** command. Enter **?** for a list of available commands (see the **ftp**(TC) manual page for descriptions). When your **ftp** command finishes processing, **ftp** displays its prompt again. You remain in **ftp** command mode until you exit **ftp** with the **quit** command.

NOTE When you log in under a certain account name with this version of **ftp**, the **ftpd** daemon checks the file */etc/shells* to make sure that the account uses a valid shell. The shell for that account must appear in */etc/shells*, or **ftpd** does not allow the user to login under *ftp*.

By default, **ftp** operates in verbose mode, displaying many messages about how it performs your file-transfer requests. If you prefer not to see these extra messages, you can toggle verbose mode off by entering the **verbose** command at the **ftp** prompt. (The **ftp** examples in this chapter were created with verbose mode turned off.)

Connecting to another machine with ftp

From the **ftp** prompt, you can connect to another machine with the **open** command followed by the name of the remote machine:

```
ftp> open warwick
Name (warwick:perry): rsimpson
Password:
ftp>
```

Press ⟨Return⟩ to use the default name shown in parentheses or enter a different user name. Then, enter the appropriate password. In this example, you can access the files belonging to *rsimpson* because you knew that person's password.

When you no longer need the remote connection, use the **ftp close** command to disconnect from the other machine and return to the **ftp** command line. End the **ftp** session with the **quit** command.

Instead of using the **open** command from within **ftp**, you can directly establish a connection to a specific machine by giving the name of the remote machine with the **ftp** command:

ftp warwick

You can log in to the remote machine and transfer files exactly as with the **open** command.

Transferring files with ftp

After you have logged in to a remote machine, you can copy a file from that machine to your machine with the **get** command. Give the pathname of the file you want to transfer from the other machine, followed by the destination on your local machine. With your connection to *warwick* already established, the following example shows how to list the directory to locate the file you want to transfer and how to copy the file *bugs.form* from the remote machine to your current directory.

```
ftp> ls /usr/local/lib/bugreport
bugs.form
formprint
forms.help
forms.msgs
      .
      .
      .

ftp> get /usr/local/lib/bugreport/bugs.form bugs.form.new
ftp>
```

Use the **put** command to copy a file from your machine to the remote machine. For example, after you have edited the bug report form, use **ftp** to copy your new version to *warwick*:

```
ftp> put bugs.form.new /usr/local/lib/bugreport
ftp>
```

Transferring files with a non-UNIX system

Different types of operating systems use different file formats. When **ftp** transfers files to or from a non-UNIX system, it transfers them in ASCII mode by default and automatically translates from one system's format to another as appropriate. Use this default ASCII mode when transferring ASCII text files with a non-UNIX system. If you want to transfer binary files with a non-UNIX system, use **ftp**'s binary mode, which copies data literally (without any translation). Switch to binary mode with the **binary** command; switch back to ASCII mode with the **ascii** command.

In general, if you transfer a file and get unexpected results, try again in the other mode.

When transferring files between 386 UNIX systems, both modes work properly for both ASCII and binary files because no translation is needed. Binary mode works faster.

Logging in automatically through the .netrc file

ftp does not understand user equivalence as defined in the */etc/hosts.equiv* and *.rhosts* files. Instead, **ftp** uses the *.netrc* file in your home directory on the local machine for login and startup information when connecting to a remote machine.

In the *.netrc* file, you can put your user name and password for each machine that you want **ftp** to log in to automatically. When you try to connect to a machine, **ftp** searches *.netrc* for an entry for that particular machine. If login information exists, **ftp** automatically supplies your user name and password (if you entered your password in the file) to the remote machine. If you did not enter your password in the file, **ftp** prompts you for it.

Because the password is given in the *.netrc* file in normal, unencrypted text, if you include your password, **ftp** requires that you set the permissions on the file so that other people do not have access to it (permissions of **400** or **600**). If the file contains your password and is not protected properly, **ftp** aborts the login process. See **chmod**(C) for more information on file permissions.

You can also define macros in the *.netrc* file. See *netrc*(SFF) for details about this file.

Using anonymous ftp

If the system administrator on the remote machine has set up a public **ftp** account, you can transfer files within certain protected directories of the **ftp** home directory without an account on a remote machine. To use this feature, log in with the user name *anonymous*. This account has no password, but it is customary to enter your user name at the password prompt.

```
ftp> open grover
Name (warwick:perry): anonymous
Password:
ftp>
```

A public account has restricted file access, but can be very useful for exchanging public software or other information.

An error message saying that the *anonymous* user is unknown usually means that the system administrator of the remote machine is not allowing public access.

ftp command options

Several command options affect how **ftp** operates:

-v specifies verbose mode (the default)

-d specifies debug mode

-i turns off prompting for multiple-file transfers (see the **ftp prompt** command)

-n prevents automatic login when connecting to a remote machine (use the **ftp user** command to log in manually)

-g turns off expansion of UNIX filename wild cards (see the **ftp glob** command)

For more information on these options, see the **ftp**(TC) manual page.

Chapter 4

Running commands remotely with rcmd

Here are some reasons to run commands on another machine:

- take advantage of some idle CPU time on the other machine
- share printers, hard disks, or other remote devices
- run software that resides on another machine
- find out information about the other machine

The **rcmd**(TC) command lets you send commands to a remote machine for execution and receive the results locally. You do not have to log in to the remote machine to use these commands. **rcmd** passes its standard input to the remotely executed command, and returns standard output and standard error from the remote command to your local system.

You cannot remotely run programs that depend on accessing the terminal directly, such as **vi**(C), because **rcmd** reads from the terminal line by line, whereas programs like **vi** use the terminal in raw mode, reading from the terminal character by character. To run a screen-oriented program on another machine, you must use **rlogin** or **telnet** to log in to that machine remotely.

Invoking rcmd

To use **rcmd**, the remote machine on which you want to run commands must be running a UNIX operating system, and you must have user or system equivalence with that machine (see the section "Establishing user equivalence" in Chapter 7 of this guide). Before executing the remote command, **rcmd** reads your shell startup file (*.cshrc* for C-shell users or *.profile* for Bourne shell users) in your home directory on the other machine if the file exists, so it can use any aliases you have defined on the other machine when executing the command.

For example, you can use **rcmd** to send files over the network to a printer connected to another UNIX system. The following example sends the file *notes* in the current directory on the local machine to the default printer on *warwick*:

cat notes | rcmd warwick lp

Using shell metacharacters

Shell metacharacters provide redirection "<" or ">", piping " | ", process control "&", and so on. You can use these special characters as part of the command to be executed remotely as well as within the local **rcmd** command syntax. When you want **rcmd** to pass the special character to the remote machine with the remote command, you must escape the special character's meaning to the local machine with a double or single quotation mark ("" or "'") or with a backslash "\". Any special character that is not escaped is interpreted on the local machine.

As an example, suppose you want to get a list of the number of blocks in all files and directories on the remote machine *warwick*, and you want to save the output in the file */tmp/warwick.du* on the remote machine. You could use the following command:

rcmd warwick du \> /tmp/warwick.du

If you leave out the backslash before the redirection symbol, the file */tmp/warwick.du* is created on the local machine.

Refer to the **csh**(C) or **sh**(C) manual page for complete information about escaping metacharacters on the command line.

rcmd command-line options

You can specify two options when invoking **rcmd**:

-l ***user*** (lowercase "L") specifies the user name under which you want to execute the command. You must have user equivalence with that user.

-n makes the command's standard input */dev/null* instead of **rcmd's** standard input. This option also prevents **rcmd** from reading and buffering standard input data.

Any option must follow the name of the remote machine on the command line. These options are described in more detail in the **rcmd**(TC) manual page.

Chapter 5
Sending mail across the network

If your system has a mail database for all users within your local network, you can send mail to another user at your site simply by specifying the person's user name with the **mail** command as described in your operating system documentation. If your system's mail database does not let you send mail to another person at your site with only the person's user name, you must follow the user name with an at-sign and the name of the person's machine:

mail ***user@machine***

Suppose you want to send mail to someone further away, perhaps to a former professor at the university you attended. To send mail across the wide-area network, you must give the **mail** program some additional information: a host name and the name of the domain in which the host exists.

A "host" is a machine connected to the network at which the person can receive mail from other machines on the network. A "domain" is a set of machines usually grouped by geographic location, organization, or type of activity (for example, EDU for educational machines, COM for machines in commercial use, UK for machines in England). Ask the person to whom you are trying to send mail for the host, domain, and user name you should use.

For example, if you want to send mail to Professor Shastraboul at the University of California at Berkeley, you would be told that the host name is *ucbvax* and the domain name is *Berkeley.EDU*. If the professor's user name is *shastra*, you can send your mail with a command like this:

mail shastra@ucbvax.Berkeley.EDU

Depending on the network connections between your site and the site to which you want to send, and depending on how the mail systems are set up at the two sites, you might not be able to use this syntax to send mail to certain remote sites. For more information, see the the mail chapter in your *UNIX System V User's Guide*.

For a description of options available with the **mail** command, see the **mail**(C) manual page.

Chapter 6

Other useful commands

You can find out certain information about other machines on the network with these commands:

finger provides information about the specified user

hostname displays the network name of the current machine

netstat displays the status of the network

nslookup queries DARPA Internet domain name servers

ruptime gives the status of machines on the local network

rwho shows who is logged in to the local network

See the networking manual pages for more information on each of these commands.

Chapter 7

Network administration

This chapter covers topics related to setting up and administering your TCP/IP network. When you installed and configured your system, many of these tasks were performed automatically by the **netconfig**(ADM) command to configure a basic networked system. For more information on **netconfig**, see the **netconfig**(ADM) manual page. If you want to customize your installation or troubleshoot your network, you should read this chapter.

Topics covered include modifying tunable kernel parameters, establishing user equivalence, setting up anonymous **ftp**, and administering pseudo ttys. In addition, if your network is not performing well, the section "Network tuning and troubleshooting" at the end of this chapter provides helpful suggestions.

Kernel configuration

Table 7-1 lists the tunable kernel parameters for this release of TCP/IP. These parameters are configured by default to work efficiently in most situations. However, if your system has an unusual configuration, you may need to tune these parameters. Please read the descriptions that follow Table 7-1 before you change any tunable kernel parameters. The following steps explain how to change the parameters:

1. Determine the name of the file to edit by looking in the second column of Table 7-1. The complete filename will be */etc/conf/pack.d/* plus the value from the table. For example, to change the value of nb_sendkeepalives, edit the file */etc/conf/pack.d/nb/space.c*. Note that the directories in *pack.d* only exist if they have been configured. For example, *ppp* will only appear if PPP has been configured using **netconfig**.

2. Make a backup copy of the file in case you need to restore the original values later.
3. Using a text editor, such as **vi**, locate the line containing the variable and change the supplied value. Where the current value is a constant that is defined in a header file, do not edit the header file. Instead, replace the constant with the new value. Where the parameter is boolean (that is, either 0 or 1), change the current value to the opposite boolean value (for example, change 0 to 1).
4. Relink the kernel using the **sysadmsh** System ⇨ Configure ⇨ Kernel ⇨ Rebuild selection. Follow the prompts; respond **y** both when asked if you want to have the new kernel boot by default and when asked if you want to rebuild the kernel environment.
5. Boot the new kernel using the **sysadmsh** System ⇨ Terminate selection to shut the system down. Press ⟨Return⟩ when the reboot message is displayed.

NOTE Randomly changing these parameters can seriously harm your system. Make absolutely sure that you understand the implications of any changes you make.

Table 7-1 Tunable kernel parameters

Parameter	File	Description	Default
ahdlcmtu	asyh/space.c	asyh MTU	296
icmp_answermask	ip/space.c	answer subnet mask requests	0
ipforwarding	ip/space.c	forward datagrams	0
ipprovcnt	ip/space.c	# of IP interfaces	16
ipsendredirects	ip/space.c	send ICMP redirects	0
NB_NNAMETAB	nb/space.c	# of NetBIOS names	64
NB_NREQ	nb/space.c	# of NetBIOS Control Blocks NCBs	64
NB_NSESSION	nb/space.c	# of NetBIOS sessions	64
nb_sendkeepalives	nb/space.c	NetBIOS level keepalives on/off	0
pppmtu	ppp/space.c	PPP MTU	296
slmtu	slip/space.c	SLIP MTU	296
subnetsarelocal	ip/space.c	treat subnets as local	1
tcpprintfs	tcp/space.c	TCP checksum errors notice	1
tcp_recvspace	tcp/space.c	TCP default receive window size	4096
tcp_round_mss	tcp/space.c	round MSS down to multiple of 1024	1
tcp_ttl	tcp/space.c	TCP default TTL	60
udp_ttl	udp/space.c	TCP default TTL	30

The following list describes how the variables affect TCP/IP and the cases in which each variable needs to be changed.

ahdlcmtu, pppmtu, slmtu
Change these variables if a machine at the other end of a serial link has a larger Maximum Transmission Unit (MTU) size which cannot be changed. The MTU is the largest amount of data that can be transferred across a given physical network. For LANs, the MTU is determined by the network hardware. For WANs, the MTU is determined by the software.

pppmtu controls the Internet Engineering Task Force Point to Point Protocol. **slmtu** controls the SLIP (Serial Line IP) protocol.

icmp_answermask
Set to 1 (true) to cause the system to respond to ICMP subnet mask requests. This variable is needed to support diskless workstations.

ipforwarding, ipsendredirects
Set these to 1 if this machine is to be used as a gateway. **ipforwarding** controls whether the system will forward packets sent to it which are destined for another system. When a packet is forwarded to another system which could have been reached directly by the originating system, an ICMP redirect message will be sent to the originating system if **ipsendredirects** is set to 1 (true).

The **netconfig**(ADM) utility also configures these values when drivers (after the first one) are added. **netconfig** edits *mtune* in the */etc/conf/cf.d* directory. This **netconfig** feature usually makes it unnecessary to change **ipforwarding** and **ipsendredirects** in the *ip/space.c* file.

ipprovcnt
Set to the number of IP interfaces you have, if you have more than network adapter cards used by IP.

NB_NSESSION, NB_NNAMETAB, NB_NREQ
These variables are respectively, the number of NetBIOS sessions, NetBIOS names and NetBIOS Control Blocks (NCBs) corresponding to the LAN Manager/X parameters **MAXSESSIONS, NNCB_NAMES** and **NNCB**. If you increase any one of those LAN Manager/X parameters above the TCP/IP default, you must change the corresponding TCP/IP parameter accordingly. For example, if you want to increase the LAN Manager/X parameter **MAXSESSIONS** to 128, you must also change **NB_NSESSION** in *nb/space.c* to 128 and relink the kernel. Note that if you change the defaults for **NB_NSESSION** in or **NB_NREQ** in nb/space.c, you should also change **NB_DFLTSSN** and**NB_DFLNCB** in /etc/default/nbconf to correspond with the new TCP/IP NetBIOS driver values.

nb_sendkeepalives
Set to 1 to turn on NetBIOS level keepalives. When turned on, NetBIOS keepalives are sent periodically on dormant NetBIOS connections. NetBIOS keepalives are independent of TCP/IP keepalives, and are useful for systems that do not use TCP/IP keepalives. This parameter is turned off by default.

subnetsarelocal
Set to 1 to request the network adapter maximum packet size on TCP/IP connections to local subnets. This is useful if the local network is made up of links which have maximum packet sizes greater than or equal to the local adapter. "ICMP Host Unreachable" is generated for local subnet routing failures. When this value is set to 0, the packet size is set to 576 bytes, as specified by RFC 1122.

tcpprintfs
If this parameter is set to 1, TCP/IP warning messages are printed to the console. If the parameter is set to 0, the messages are not printed anywhere.

tcp_recvspace
This parameter affects the size of the TCP receive window. If you need a faster TCP data-receive rate, increase **tcp_recvspace**. For example, increase this parameter if your machine receives a lot of data from remote sites.

The parameter is set to **TCPWINDOW**. Change **TCPWINDOW**'s value by changing the following line in */etc/conf/pack.d/tcp/space.c*:

```
#define TCPWINDOW 4*1024
```

We recommend that you initially change the value to 24*1024. You cannot increase the value beyond 64*1024.

Increasing the receive window size can sometimes result in exhausting other resources. After you increase the **TCPWINDOW** value, monitor your STREAMS buffers, network card statistics, and protocol-specific statistics. See "Network tuning and troubleshooting" (at the end of this chapter) and the **netstat**(TC) manual page for information on monitoring these resources.

tcp_round_mss
If this parameter is set to 1, the TCP/IP Maximum Session Size is rounded down to the nearest multiple of 1024. Setting this parameter to 0 prevents the round-down, letting you more effectively use the data portion of a TCP segment. This potentially decreases the number of segments you need for transferring a given amount of data.

tcp_ttl, udp_ttl
Increase these parameters if you have an Internet connection with many hops.

Setting interface parameters

All network interface drivers, including the loopback interface, require that their host addresses be defined at boot time. This is done with **ifconfig**(ADMN) commands included in the */etc/tcp* shell script. These commands are initially set up based on information you supply during configuration with **netconfig**(ADM).

ifconfig(ADMN) also sets the netmask, broadcast address, and IP address at boot time. If you want to change any of this information, you can edit */etc/tcp* (after making a backup copy of the file), then reboot your system. Instances in which you may want to do so include:

- changing the netmask and broadcast address to make provisions for a subnetwork
- changing the broadcast address scheme from all ones to all zeroes, or vice versa
- changing the IP address of the interface

Creating a subnetwork

In TCP/IP, the DARPA Internet support includes the concept of the subnetwork. This is a mechanism that enables several local networks to appear as a single Internet network to off-site hosts. Subnetworks are useful because they allow a site to hide the local topology, requiring only a single route in external gateways. This also means that local network numbers may be locally administered.

To set up local subnetworks, you first need to know how much of the available address space is to be partitioned. The term "address" is used here to mean the Internet host part of the 32-bit address. Sites with a class A network number have a 24-bit address space with which to work, sites with a class B network number have a 16-bit address space; and sites with a class C network number have an 8-bit address space. To define local subnets you must steal some bits from the local host address space for use in extending the network portion of the Internet address.

This reinterpretation of Internet addresses is done only for local networks. It is not visible to off-site hosts. For example, if your site has a class B network number, hosts on this network have an Internet address that contains the network number, 16 bits, and the host number, another 16 bits. To define 254 local subnets, each possessing at most 255 hosts, 8 bits may be taken from the local part to be used for the subnetwork ID. (The use of subnets 0 and all-1's, 255 in this example, is discouraged to avoid confusion about broadcast

addresses.) New network numbers are then constructed by concatenating the original 16-bit network number with the extra 8 bits containing the local sub-network number.

Network servers

In the UNIX system, most of the server programs are started by a super server, called the "Internet daemon". The Internet daemon, */etc/inetd*, acts as a master server for programs specified in its configuration file, */etc/inetd.conf*, listening for service requests for these servers, and starting up the appropriate program whenever a request is received. The configuration file includes lines containing:

- a service name (as found in */etc/services*),
- the type of socket the server expects (for example, stream or dgram),
- the protocol used with the socket (as found in */etc/protocols*),
- whether to wait for each server to complete before starting up another,
- the user name under which the server should run,
- the server program's name, and
- up to five arguments to pass to the server program

Some trivial services are implemented internally in *inetd*(SFF), and their servers are listed as internal. For example, an entry for the file-transfer protocol server would appear as:

```
ftp   stream   tcp   nowait   root   /etc/ftpd   ftpd
```

Consult *inetd*(SFF) for more details on the format of the configuration file and the operation of the Internet daemon.

Network databases

Several data files are used by the network library routines and server programs. Most of these files are host independent and updated only rarely. Table 7-2 lists the data files used:

Table 7-2 Network database files

File	Manual reference	Use
/etc/hosts	*hosts* (SFF)	host names
/etc/networks	*networks* (SFF)	network names
/etc/services	*services* (SFF)	list of known services
/etc/protocols	*protocols* (SFF)	protocol names
/etc/hosts.equiv	**rshd** (ADMN)	list of "trusted" hosts
/etc/ftpusers	**ftpd** (ADMN)	list of "unwelcome" **ftp** users
/etc/inetd.conf	**inetd** (ADMN)	list of servers started by **inetd**

The files distributed are set up for ARPANET or other Internet hosts. Local networks and hosts should be added to describe the local configuration. Network numbers must be chosen for each Ethernet. For sites not connected to the Internet, these can be chosen more or less arbitrarily; otherwise, the normal channels should be used for allocation of network numbers.

Establishing user equivalence

User equivalence allows a user on one machine to use **rlogin** to log into an equivalent account on another machine without entering a password. You need to modify several files to establish user equivalence: */etc/hosts.equiv*, *.rhosts*, and */etc/passwd*.

To do so, ensure that entries for the affected user exist on both machines in the following files:

- *.rhosts* and */etc/passwd*, or
- */etc/hosts.equiv* and */etc/passwd*

In both cases, */etc/passwd* must contain an entry for the user name from the remote machine. If you add an entry to *.rhosts* for a particular account, then user equivalence is established for that account only. If you add an entry in */etc/hosts.equiv* for a host name and an account on that host, then that account has user equivalence for any account (except *root*). If the entry in */etc/hosts.equiv* has only the remote host name, then any user on that host has user equivalence for all local accounts (except *root*). Such a host is considered a "trusted host."

NOTE Entries in */etc/hosts.equiv* can create large holes in system security. Be sparing in their use. In most circumstances, it is unwise to create entries that allow all users on remote machines to access all accounts on your local machine.

For example, suppose you have an account under the user name *test1* on machine *admin*. You want to establish user equivalence on the remote machine *systemb*. The administrator for the machine *systemb* must create an account for user name *test1*. They must also include the following entry in the file */etc/hosts.equiv* on *systemb*:

```
admin  test1
```

This gives user equivalence for all accounts except *root* to user *test1* on the machine *systemb*. Suppose that test1 really only needed access to the account *testb* on *systemb*. Then it would be better to remove the above entry from */etc/hosts.equiv* on *systemb* and use the following entry in the file *.rhosts* in the home directory for *testb*:

```
admin  test1
```

Note that entries for *.rhosts* must include both the system name and the account name. The file */etc/hosts.equiv* does allow entries for the system name only, as discussed earlier.

If there are entries in both *.rhosts* and */etc/hosts.equiv* for the same machine or machine/account combination, then the entry from */etc/hosts.equiv* determines the extent of user equivalence.

Setting up anonymous ftp

The **ftp** server included in the system provides support for an anonymous **ftp** account. Because of the inherent security problems with such a facility, you should read this section carefully if you want to provide such a service.

An anonymous account is enabled by creating a user called *ftp*. When a client uses the anonymous account, a **chroot**(ADM) system call is performed by the server to restrict the client from moving outside that part of the filesystem where the *ftp* home directory is located. Because a **chroot** call is used, certain programs and files used by the server process must be placed in the *ftp* home directory. Further, you must be sure that all directories and executable images are unwritable. The following directory setup is recommended:

```
# csh
# cd ~ftp
# chmod 555 .; chown ftp .; chgrp ftp .
# mkdir bin etc pub shlib dev
# chown root bin etc shlib dev
# chmod 555 bin etc shlib dev
# chown ftp pub
# chmod 777 pub
# cd bin
# cp /bin/sh /bin/ls /bin/pwd .
# chmod 111 sh ls pwd
# cd ../etc
# cp /etc/passwd /etc/group .
# chmod 444 passwd group
# cd ../shlib
# cp /shlib/libc_s .
# cd ..
# find /dev/socksys -print | cpio -dumpv .
```

When local users want to place files in the anonymous area, they must place them in a subdirectory. In the setup here, the directory *~ftp/pub* is used.

Another issue to consider is the */etc/passwd* file placed here. It can be copied by users who use the anonymous account. They can then try to break the passwords of users on your machine for further access. A good choice of users to include in this copy might be *root*, *daemon*, *uucp*, and the *ftp* user.

Aside from the problems of directory modes and such, the **ftp** server provides a loophole for interlopers if certain user accounts are allowed. The file */etc/ftpusers* is checked on each connection. If the requested user name is located in the file, the request for service is denied. It is suggested that this file contain at least the following names:

```
uucp
root
```

Accounts with nonstandard shells should be listed in this file. Accounts without passwords need not be listed in this file; the **ftp** server does not service these users.

Administering pseudo ttys

When you install TCP/IP, the **netconfig** command allows you to modify the number of pseudo ttys configured on your system. These pseudo ttys allow outside machines to use **telnet** or **rlogin** to access TCP/IP on your machine. If you want to remove these pseudo ttys, or add more to your system, after installing and configuring TCP/IP, use the **sysadmsh**(ADM) command:

1. Log in to the system as *root* and bring the system into system maintenance mode.
2. Enter the following command at your prompt and press ⟨Return⟩:

   sysadmsh

3. Select the following: System ➪ Hardware
4. Select Pseudo ttys from the point-and-pick list.
5. Enter **1** to add pseudo ttys, or **2** to remove pseudo ttys. If you are adding pseudo ttys, continue with the next step. If you are removing pseudo ttys, skip to step 7.
6. Enter a number of pseudo ttys to create and press ⟨Return⟩. If the number is greater than the current system limit, you are asked if you want to increase the limit. Enter **y** to increase the limit or **n** to cancel the increase.

   Skip to step 8.
7. You return to the main menu. Enter **q** to quit.
8. Enter a number of pseudo ttys to remove and press ⟨Return⟩.
9. You are prompted to relink the kernel. Do so by entering **y** at the prompt.

   NOTE If you do not relink the kernel now, the changes you made do not take place, and you return to **sysadmsh**. You must then relink at a later time to effect the changes.

10. The following message appears:

```
The UNIX Operating System will now be rebuilt.
This will take a few minutes.  Please wait.
Root for this system build is /.
Do you want this kernel to boot by default? (y/n)
```

Answer **y**.

11. The following prompt appears:

```
Do you want the kernel environment rebuilt? (y/n)
```

    Answer **y**. A message appears confirming that the kernel is linked and installed.

12. Exit **sysadmsh** and reboot your system by entering the following command:

    init 6

Network tuning and troubleshooting

It is likely that from time to time you will encounter problems using your network. The first thing to do is check your network connections. On networks such as the Ethernet a loose cable tap or poorly placed power cable can result in severely deteriorated service. The **ping**(ADMN) command is particularly useful for confirming the existence of network connections. If there is no hardware problem, check next for routing problems and addressing problems.

The **netstat**(TC) program can also be helpful in tracking down hardware malfunctions. In particular, look at the **-i** and **-s** options in the manual page. The **netstat**(TC) program also shows detailed information about network behavior. Examples of **netstat** displays appear later in this chapter.

If you think a communication protocol problem exists, consult the protocol specifications and attempt to isolate the problem in a packet trace. The **SO_DEBUG** option can be supplied before establishing a connection on a socket, in which case the system traces all traffic and internal actions (such as timers expiring) in a circular trace buffer. This buffer can then be printed out with the **trpt**(ADMN) program. Most of the servers distributed with the system accept a **-d** option forcing all sockets to be created with debugging turned on. Consult the appropriate manual pages for more information.

STREAMS tuning

If your system does not have enough STREAMS buffers it will lose connections for no reason, hang on processes that communicate over the network, cause programs that communicate over the network to malfunction, and hang on any process that deals with terminal I/O.

Run **netstat -m** at the prompt to display the current STREAMS buffer use. The FAIL column should have a 0 in every row. If failures are listed in this column, increase the corresponding STREAMS buffers. For example, if you have 40 failures in the **NBLK512** row, increase the kernel parameter **NBLK512.** Start by increasing the number of buffers by 50%. If you still have failures, increase **NBLK512** again by 50% of the original value until there are no failures.

To increase STREAMS buffers, use **sysadmsh** and select the following:

System ➪ Configure ➪ Kernel ➪ Parameters

Choose the Streams Data option. You are prompted to enter a value for all STREAMS parameters. Press ⟨Return⟩ for values that you do not want to change. Enter the new value for the STREAMS buffer that you want to increase. You can also use the UNIX Link Kit **configure**(ADM) command to increase STREAMS buffer resources.

Active connections display

The active connections display is the default display of the **netstat**(TC) command. It displays a line of information for each active connection on the local machine under the following heading.

netstat -a

Active Internet connections (including servers) are as follows:

```
scobox$ netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address       Foreign Address    (state)
ip         0      0       *.*                *.*
tcp        0      0       scobox.telnet      scoter.2460     ESTABLISHED
tcp        0      0       *.smtp             *.*             LISTEN
tcp        0      0       *.1024             *.*             LISTEN
tcp        0      0       *.sunrpc           *.*             LISTEN
tcp        0      0       *.chargen          *.*             LISTEN
tcp        0      0       *.daytime          *.*             LISTEN
tcp        0      0       *.time             *.*             LISTEN
tcp        0      0       *.domain           *.*             LISTEN
tcp        0      0       *.finger           *.*             LISTEN
tcp        0      0       *.exec             *.*             LISTEN
tcp        0      0       *.ftp              *.*             LISTEN
tcp        0      0       *.telnet           *.*             LISTEN
tcp        0      0       *.login            *.*             LISTEN
tcp        0      0       *.shell            *.*             LISTEN
tcp        0      0       scobox.listen      *.*             LISTEN
tcp        0      0       scobox.nterm       *.*             LISTEN
tcp        0      0       *.*                *.*             CLOSED
udp        0      0       *.1035             *.*
udp        0      0       *.1034             *.*
udp        0      0       *.1033             *.*
udp        0      0       *.1032             *.*
udp        0      0       *.2049             *.*
udp        0      0       *.1028             *.*
udp        0      0       *.sunrpc           *.*
udp        0      0       scobox.domain      *.*
udp        0      0       localhost.domain   *.*
scobox$
```

Descriptions of the display headings

Proto	protocol used in the connection
Recv-Q	receive queue. This is the number of received characters (bytes) of data waiting to be processed.
Send-Q	send queue. This is the number of characters (bytes) of data waiting to be transmitted.
Local Address	port number of the local connection, displayed symbolically. The port numbers are taken from the */etc/services* file.
Foreign Address	port number of the remote connection, displayed symbolically. The port numbers are taken from the */etc/services* file.
State	current state of the connection. Each protocol has its own set of states. For the protocol-dependent states that can be displayed, see the appropriate protocol specification.

Interfaces

The interface display describes activities on all the local machine's interfaces to the net, in the form of a table of cumulative statistics. This display is available through **netstat** with the **-i** option.

netstat -i

```
scobox$ netstat -i
Name Mtu  Network     Address   Ipkts   Ierrs   Opkts   Oerrs   Collis
en0  1500 sco-eng-ne  scobox       No Statistics Available
e3B0 1500 128.174.14  128.174.14.10     0       0       0       0
lo0  2048 loopback    localhost189      0       189     0       0
scobox$
```

Descriptions of the display headings

Each interface is described by a line with the following headings:

Name	name of the network interface. For example, *en0* is the name of the first Ethernet interface board.
Mtu	maximum transmission unit (in bytes). This is the largest size permitted for any single packet sent through this interface.
Network	name of the network address of the interface as given in */etc/networks*

Address	name of the machine address of the interface in */etc/hosts*
Ipkts	input packets. This is the number of packets received on the interface.
Ierrs	input errors. This is the number of errors detected in packets of data received on this interface.
Opkts	output packets. This is the number of packets transmitted on the interface.
Oerrs	output errors. This is the number of errors detected and corrected in packets of data transmitted on this interface.
Collis	collisions that occurred on the network

Routing tables

The Routing Table display provides information about the usage of each route you have configured. A route consists of a destination host or network and a network interface used to exchange packets. Direct routes are created for each interface attached to the local host.

netstat -r

```
scobox$ netstat -r
Routing tables
Destination     Gateway         Flags   Refcnt  Use     Interface
localhost       localhost       UH      4       0       lo0
sco-eng-net     scobox          U       4       537     en0
128.174.14      128.174.14.1    U       0       0       e3B0
128.174         scoffle         UG      0       0       en0
scobox$
```

Descriptions of the display headings

The information displayed for each route is as follows.

Destination	network or machine to which this route allows you to connect
Gateway	name of the gateway you configured for this route. If you are directly connected, this is a local address. Otherwise, it is the name of the machine through which packets must be routed.
Flags	state of the route. Valid states are: U up G a route to a gateway N a route to a network H a route to a host
Refcnt	current number of active connections using the route. Connection-oriented protocols normally hold on to a single route for the duration of the connection, while connectionless protocols obtain a route, then discard it as needed.
Use	current number of packets sent using this route
Interface	name of the physical network interface used to begin the route

Statistics display

The Protocol Statistics display provides protocol-specific errors. The errors in the display are grouped under headings for each higher-level protocol in your system. The headings are protocol-specific.

- Internet Protocol (ip)
- Internet Control Message Protocol (icmp)
- Transmission Control Protocol (tcp)
- User Datagram Protocol (udp)

netstat -s

```
ip:
        3209 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        0 fragments received
        0 fragments dropped (dup or out of space)
        0 fragments dropped after timeout
        0 packets forwarded
        0 packets not forwardable
        0 redirects sent
icmp:
        1 call to icmp_error
        0 errors not generated because old message was icmp
        Output histogram:
                destination unreachable: 1
        0 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                destination unreachable: 640
        0 message responses generated
tcp:
        348 packets sent
                202 data packets (3661 bytes)
                0 data packets (0 bytes) retransmitted
                101 ack-only packets (60 delayed)
                0 URG only packets
                0 window probe packets
                0 window update packets
                45 control packets
```

(Continued on next page.)

(Continued)

```
        411 packets received
                233 acks (for 3654 bytes)
                19 duplicate acks
                0 acks for unsent data
                200 packets (1677 bytes) received in-sequence
                0 completely duplicate packets (0 bytes)
                0 packets with some dup. data (0 bytes duped)
                9 out-of-order packets (0 bytes)
                0 packets (0 bytes) of data after window
                0 window probes
                0 window update packets
                0 packets received after close
                0 discarded for bad checksums
                0 discarded for bad header offset fields
                0 discarded because packet too short
        25 connection requests
        12 connection accepts
        21 connections established (including accepts)
        72 connections closed (including 0 drops)
        16 embryonic connections dropped
        233 segments updated rtt (of 259 attempts)
        0 retransmit timeouts
                0 connections dropped by rexmit timeout

        0 persist timeouts
        0 keepalive timeouts
                0 keepalive probes sent
                0 connections dropped by keepalive
        0 connections lingered
                0 linger timers expired
                0 linger timers cancelled
                0 linger timers aborted by signal
udp:
        0 incomplete headers
        0 bad data length fields
        0 bad checksums
```

Chapter 8

Administering serial line communications

This chapter describes two Internet Protocol (IP) networking products that operate over serial lines: SLIP (Serial Line Internet Protocol) and PPP (Point-to-Point Protocol). SLIP and PPP let two computers transfer information, using the serial port in place of a network card and a standard serial cable in place of an Ethernet cable. Both SLIP and PPP are commonly used on dedicated serial links. They are useful for allowing mixes of hosts and gateways to communicate with one another (host-host, host-gateway, and gateway-gateway are all common SLIP and PPP network configurations). You can operate SLIP and PPP at any speed from 1200 baud to 19.2K baud, as long as your serial port supports the speed.

The first section of this chapter covers SLIP, and the second covers PPP. Each section describes the protocol's features, explains the configuration procedure, and offers troubleshooting suggestions.

Administering SLIP

SLIP functionality is limited to defining a sequence of characters that frame IP packets (or *datagrams*) on a serial line. SLIP provides no addressing, packet type identification, or error detection or correction, and allows only asynchronous transmission. SLIP does offer a compression mechanism for packet headers. PPP, described later in this chapter, provides some enhancements to SLIP's capabilities.

The TCP/IP Runtime system includes SLIP in the basic TCP/IP installation package, so you install it automatically when you install TCP/IP.

Configuring a SLIP connection

This section begins with information that you need before you begin configuring SLIP. The section then describes step-by-step procedures for configuring three types of SLIP lines: a direct SLIP connection, a dialup SLIP connection, and a SLIP/Ethernet or SLIP/Token-Ring gateway.

Preparing to configure a SLIP connection

SCO TCP/IP supports up to four SLIP lines per machine. Each SLIP line requires a unique network address and hostname, making it a separate subnet. For example, a hostname for the first SLIP line on a machine called *opal* might be *opal_slip1*, the second *opal_slip2*, and so forth. For more information on network addresses, see the section on IP addresses in the introduction to this guide.

Chapter 1 of this guide contains the information that you need to configure a SLIP connection. If you read this material before you configure the SLIP driver, the configuration process flows more smoothly than if you attempt to answer prompts during configuration.

Information you need to know includes:

- name of the tty line you plan to use
- baud rate at which you are operating SLIP
- IP address (or Internet address) of your system
- IP address of the target system to which you are connecting the SLIP line
- hostname of the target system to which you are connecting the SLIP line

Configuring a direct SLIP connection

This section describes how to use **netconfig**(ADMN) to configure a direct SLIP connection between a machine named *emerald* and a machine named *ruby*. The following procedure assumes that the machines are connected through their **COM1** serial ports using a null modem cable, and both machines have TCP/IP installed. After you enter each selected value, press ⟨Return⟩ to complete your response:

1. Boot the machine *emerald* in System Maintenance mode by entering the *root* password at the following prompt:

```
Type CONTROL-D to proceed with normal startup (or enter root
passwd for System Maintenance Mode):
```

2. Enter the following command:

   netconfig

3. At the Available options menu, enter the number that corresponds to Add a chain. A *chain,* in **netconfig** terms, is a communications configuration that binds networking products together as though by a chain.
4. From the list of available top-level products, enter the number that corresponds to sco_tcp. (SLIP is a TCP/IP product.)
5. At the prompt `Select next level of chain to Add or q to quit`, enter the number that corresponds to sl0.
6. When a prompt asks you to confirm the selected product chain, enter **y** if the displayed product chain is correct. If the chain is incorrect, enter **n** to return to the **netconfig** main menu. You can then either quit **netconfig** or add a different chain of products.
7. At the prompt `Enter tty line to be used for slip`, enter the name of the tty line in this format: **tty1a**.
8. At the prompt `Enter Baud rate for tty1a`, enter the baud rate. The default baud rate is 9600.
9. At the prompt `Enter the internet address of this interface`, enter the IP address for your system (*emerald*) in this format: ***nnn.nnn.nnn.nnn.*** See the introduction to this guide for a discussion of IP addresses.
10. At the prompt `Enter the internet address of the destination interface:`, enter the address for the target system (*ruby*) in this format: ***nnn.nnn.nnn.nnn.***
11. At the prompt `Enter hostname of the destination interface:`, enter the name of the target system; in this example, **ruby**.
12. The prompt `Are these values correct? (y/n)`, gives you a chance to change your choices. If you wish to alter any of the values, enter **n**, then follow the screen prompts. If the values are correct, enter **y**.

    The **netconfig** utility automatically adds *ruby*'s information to *emerald*'s */etc/hosts* file.
13. At the prompt `Configure gateway?`, enter **n**, unless you plan to establish a SLIP/Ethernet or SLIP/Token-Ring gateway as described later in this chapter.
14. When **netconfig** prompts you to add or remove pseudo ttys, it displays the current number of pseudo ttys already configured on the system. If this number is correct, enter **q** to quit the pseudo tty configuration screen. If you want to add or remove pseudo ttys, enter the number corresponding

to the desired option and follow the prompts. For more information on pseudo ttys, see the chapter "TCP/IP network administration" earlier in this document.

15. When you again see the Available options menu, enter **q**.
16. You see a prompt to relink the kernel. Enter **y** for the first SLIP configuration. Subsequent SLIP configurations on the same machine do not require relinking the kernel.
17. During kernel relinking, enter **y** to have this kernel boot by default, and **y** again to have the kernel environment rebuilt. This procedure takes a few minutes. When the rebuild is complete, you return to the command line.
18. Disable both the modem and non-modem control ports corresponding to the tty device being used for SLIP. (The ports might already be disabled.) For example, use the following commands for **COM1**:

    disable tty1a
    disable tty1A

19. If you want to use header compression, edit the */etc/tcp* script, and add the **+c** flag (to turn on compression mode) or the **+e** flag (to turn on compression detection) to the **slattach**(ADMN) command. Using header compression lets you make the most efficient use of the serial line. See the **slattach**(ADMN) manual page for more information.
20. Reboot *emerald*. On booting up with the new kernel, TCP/IP should start up successfully using the SLIP interface.
21. Use the same procedure to configure the SLIP line on *ruby*, making the interface address for *ruby*'s SLIP line match the destination address for *emerald*'s SLIP line, and making the destination address for *ruby*'s SLIP line match the interface address for *emerald*'s SLIP line.

This completes the configuration of the SLIP interface on both *emerald* and *ruby*. If you want to configure more than one SLIP line on a machine, repeat the **netconfig** procedure for each line you wish to configure, selecting the option for sl1, and so on. Make sure that each SLIP line is a separate subnet (has a unique network address and hostname).

Configuring a dialup SLIP connection

SCO TCP/IP does not directly support dialup SLIP lines. If PPP is available on both machines, configure a dialup connection using PPP rather than SLIP. However, if you cannot use PPP, you can configure dialup SLIP lines using the workaround described in this section. This workaround has not yet been tested on the current version of TCP/IP.

NOTE Dialup SLIP uses one of the dialer programs from the operating system's UUCP package; for example **/usr/lib/uucp/dialHA12** for a Hayes™ modem at 1200 baud. If UUCP and the appropriate dialer program are not installed, use **custom**(ADM) to install them.

The following procedure describes how to configure a dialup SLIP connection between a machine named *jade* and a machine named *diamond* across *tty1a* at 1200 baud. The procedure assumes that both machines have TCP/IP installed and SLIP configured.

1. Execute the following commands for *diamond* at the super-user shell prompt or through the */etc/tcp* script:

```
(stty 1200; echo "ATE0\r" > /dev/tty1a) < /dev/tty1a
slattach /dev/tty1a diamond_IP_address jade_IP_address 1200
```

   The commands on the first line put the modem into non-echo mode, and the second line is the **slattach**(ADMN) command. *diamond* must execute the **slattach** command before *jade* connects over the dialup SLIP connection.

   For information on selecting IP addresses for *diamond* and *jade*, see the introduction to this guide.

2. Execute the following commands for *jade* at the super-user shell prompt or through a shell script:

```
(stty 1200; echo "ATE0\r" > /dev/tty1a) < /dev/tty1a
/usr/lib/uucp/dialer_program /dev/tty1a 5551212 1200

if [ $? ]
then
slattach /dev/tty1a jade_IP_address diamond_IP_address 1200
else
echo "Error dialing\n"
fi
```

   The commands on the first line put the modem into non-echo mode, and the second line uses a UUCP dialer program to dial *diamond* at the phone number 555-1212. The subsequent lines execute **slattach** if the dialer program reports success. Make sure that *jade* dials *diamond* before issuing **slattach**.

Double-check the following points when you configure a dialup SLIP:

- each SLIP line is a separate subnet (has a unique network address and hostname)
- the **slattach** command specifies the same, supported baud rate for both machines
- both modems are using the same error protocols, data compression types, and other configurable features
- you use dedicated tty lines for the SLIP connection
- you use a non-modem control file for the device special file (for example, use */dev/tty1a* instead of */dev/tty1A* for COM1)
- the modem and non-modem control ports are disabled
- the modem is attached to the same port that the **slattach** command specifies
- the **routed**(ADMN) daemon starts after **slattach**. If **routed** starts before **slattach**, the following message appears on the console at roughly one-minute intervals:

```
routed: packet from unknown router net_address
```

If you see this message, kill **routed**, and then restart it.

Configuring a SLIP/Ethernet or SLIP/Token-Ring gateway

This section explains how to set up a machine as a gateway between a SLIP network and an Ethernet network. Use the same procedure to configure a SLIP/Token-Ring gateway, a PPP/Ethernet gateway, or a PPP/Token-Ring gateway.

A SLIP/Ethernet gateway must connect two distinct networks; that is, the network address and hostname for the SLIP side must be different from those of the Ethernet side. For a complete discussion on network addresses and host addresses, see the section on IP addresses in the introduction to this guide.

This procedure describes a SLIP/Ethernet gateway for which the machine *emerald* has the serial interface for the SLIP connection, the machine *ruby* has the Ethernet card for the Ethernet connection, and the machine *opal* is the gateway system. Both networks are configured as Class C networks. (For a discussion of network classes, see the introduction to this guide.)

This configuration procedure assumes that *opal* has TCP/IP installed, but neither the SLIP driver nor the Ethernet driver is configured:

1. Configure the SLIP interface on *opal*, using the steps outlined for a direct connection or for a dialup connection. Enter **y** at the prompt `Configure gateway?`.
2. Run **netconfig** to configure the Ethernet card on *opal*, following the instructions in the *SCO TCP/IP Release and Installation Notes*. Enter **y** at the prompt `Configure gateway?`.
3. Edit the */etc/networks* file on *opal* to include the hostname and network address for each network as follows:

   ```
   ruby_ethernet    ruby_net_address
   emerald_slip     emerald_net_address
   ```

4. If */etc/if.ignore* contains a line for sl0, remove the line.
5. Reboot *opal*.
6. Use **netstat -i** to check that both interfaces are configured. These three entries should appear in the first column: `loopback`, `sl0`, and an Ethernet driver such as `wdn0` or `e3B0`. Each entry should show non-zero values for incoming packets (Ipkts) and outgoing packets (Opkts).

Removing SLIP

Remove SLIP with **netconfig**, then restore the files that you changed to reflect the SLIP line.

1. At the Available options menu, enter the number corresponding to Remove a chain, and press ⟨Return⟩.
2. Follow the directions in the rest of the **netconfig** screen prompts.
3. Edit the */etc/hosts* file on each machine and remove the entry associated with each SLIP line.
4. If you are removing a SLIP/Ethernet or SLIP/Token-Ring gateway, remove the network entries from */etc/networks*.

Troubleshooting SLIP configurations

This section describes many of the common error messages generated from **ping**(ADMN), **telnet**(TC), or **rlogin**(TC). For more information on troubleshooting your network connections, see the chapter on network administration earlier in this guide.

Common problems with SLIP

If you encounter a problem with SLIP, check to make sure the following are true:

- **slattach** is running at both ends of the SLIP connection
- the source address of the first machine matches the destination address of the second machine, and the destination address of the first machine matches the source address of the second machine
- the machines at both ends of the connection expect to send and receive packets with compressed headers, or neither machine expects to send or receive compressed headers (that is, both machines agree about header compression)
- the machines at both ends of the connection are using the same baud rate
- the modems at both ends of the connection are using the same protocols

Verifying serial cable connectivity

To verify that your problem is not a serial cable connectivity problem, follow this procedure on both machines to which the serial cable connects:

1. While logged on as *root*, bring the system to single-user mode.
2. Edit the file */usr/lib/uucp/Devices* to uncomment the following lines, by removing the number sign from each line:

```
# Direct ttyla - 9600 direct
# Direct tty2a - 9600 direct
```

3. After you perform the previous steps on both machines, enter the following command on the source machine: **cu -l ttyla dir.**

   If the screen displays `connected`, enter some characters. They should appear on the destination machine only. If they do, your serial line is sound. Type **~.** at the beginning of a line to exit the **cu** program.

   If there is no response, the problem probably lies with the cable. Check the cable's connections, or try another cable. If an error message appears, see the chapter "Building a remote network with UUCP" in the *SCO UNIX System Administrator's Guide.*

Troubleshooting problems with ping

Test that the local and remote interfaces are functional and responding by booting the system in multi-user mode and executing the following **ping** commands:

ping ***local_hostname***
ping ***remote_host_IP_address***
ping ***remote_hostname***

If all these commands successfully respond to transmitted packets, the connection is functional and you can try other TCP/IP commands, such as **telnet** and **rlogin.**

The following error message from **ping** indicates that the network address of the remote host is different from that of the local host:

```
ping send: Network is unreachable
```

Make sure that the network portion of the remote host's IP address is correct. (See the introduction to this guide for a discussion of the network portion of the IP address.) For machines on the same network, the network part of the IP addresses for the local host and the remote host should match. If the machines are not on the same network, you need to configure a gateway to route packets between the two networks.

The following error message from **ping** indicates that the ***remote_hostname*** used on the **ping** command line does not have an entry in */etc/hosts*:

```
ping: unknown host remote_hostname
```

Edit the */etc/hosts* file on the local host to include an entry for the remote host.
If **ping** hangs with no response:

1. Make sure that the IP address used on the command line is correct. It should be the value assigned to the machine connected to the other end of the direct or modem line.
2. Use the **ifconfig**(ADMN) command as follows to check the local SLIP interface:

   ifconfig sl*n*

   This command should return an output similar to the following:

   ```
   sln: flags=51<UP,POINTOPOINT,RUNNING>
        inet source address-->destination address netmask ffff0000
   ```

   Check that the source and destination addresses are correct and have corresponding entries on the system connected to the other end of the line.
3. Check the netmasks of both machines. An incorrect netmask can render a host or network unreachable. For more information on netmasks, see the introduction to this guide.

Troubleshooting problems with rlogin or telnet

If **ping** works correctly but **rlogin**(TC) or **telnet**(TC) fails, consult the appropriate manual page to make sure that your command syntax is correct. If the syntax is correct, verify that your */etc/hosts* file has an entry for the target host.

The following **telnet** or **rlogin** error message indicates either that there are no more pseudo ttys available to make additional connections, or that the terminal database does not contain entries for the */dev/ttyp*** the system is attempting to use:

```
Cannot obtain database information on this terminal.
Connection closed.
```

For information on gaining access to additional pseudo ttys, see the section on administering pseudo ttys in the chapter "Network administration" earlier in this guide.

More SLIP information

The following manual pages provide information about SLIP:

Manual page	Description
slip(ADMP)	SLIP implementation
slattach(ADMN)	the command for assigning a tty line to a network interface
strcf(SFF)	STREAMS configuration file format

For more information about how SLIP is designed and implemented, refer to RFC 1055 by J. Romkey. For information about compressing SLIP headers, refer to RFC 1144 by Van Jacobson. You can get these documents from Jon Postel, RFC Editor, USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292-6695.

Administering PPP

Some of the enhancements that Point-to-Point Protocol (PPP) provides to SLIP's capabilities include error detection, process authentication, and the ability to send data asynchronously or synchronously.

PPP's features include the following:

- dynamic connect and disconnect service to reduce phone line costs during idle time. When the phone line disconnects, the process using PPP continues in service without receiving a carrier hangup signal. When the process transmits or receives data after idling, the phone link is automatically re-established.

- simultaneous serial line sharing with UUCP and **cu**(C) services
- error detection of transmitted data
- adaptability to high-speed links such as T1, cellular phones, and dial-in services
- ease of setup with the **netconfig**(ADM) administrative utility

The TCP/IP Runtime system supplies PPP as a separate installation package that you can install automatically with the rest of the TCP/IP product or separately.

NOTE PPP is supplied with device drivers for use with IP datagrams and for asynchronous High-level Data Link Control (HDLC) serial data transmission. PPP requires customized device drivers for synchronous transmission. Information on creating device drivers is available in the *SCO UNIX Device Driver Writer's Guide*. Note that creating a device driver is a complex and difficult task requiring specialized programming skills and a thorough knowledge of UNIX system internals.

PPP compared to SLIP

PPP was created to enhance the capabilities of the Serial Line Internet Protocol (SLIP) discussed earlier in this chapter. The following table compares these two products:

SLIP	PPP
Asynchronous transmission only	Asynchronous or synchronous
Internet Protocol only	Multiple protocol stack capability
Serial line always in use	Dynamic connect and disconnect optimizes line use during idles
Exclusive line use	Can be shared with UUCP and **cu**
No error detection of datagrams	Built-in error detection

Configuring PPP

This section begins with information that you need before you begin configuring PPP, then describes the **netconfig** procedure for configuring PPP. Finally, the section explains how to edit the UUCP *Systems* and *Devices* files (and the *Dialers* file, if the PPP line operates over modems) to configure the software for a PPP serial line.

You can edit the *Systems*, *Devices*, and *Dialers* files before or after running **netconfig**. Editing them before running **netconfig** is better, so that you can verify connectivity.

Preparing to configure PPP

Before configuring PPP, make sure that UUCP is installed on your system. Refer to the chapter "Building a remote network with UUCP" the *System Administrator's Guide* supplied with your system for information about how to set up UUCP.

SCO TCP/IP supports up to four PPP lines per machine. Each PPP line requires a unique network address and hostname, making it a separate subnet. For example, a hostname for the first PPP line on a machine called *opal* might be *opal_ppp1*, the second *opal_ppp2*, and so forth. Network addresses are discussed in the section on IP addresses in the introduction to this guide.

Chapter 1 of this guide contains contains the information that you need to configure a PPP connection. If you read this material before you configure the SLIP driver, the configuration process flows more smoothly than if you attempt to answer prompts during configuration.

Information you need to know includes:

- IP address (or Internet address) of your system
- IP address of the target system to which the PPP line is being connected
- netmask for this interface

For more information on IP addresses and netmasks, see the introduction to this guide.

Configuring PPP with netconfig

This section describes how to use **netconfig**(ADMN) to configure PPP. The **netconfig** utility can configure several PPP serial lines. Even though PPP permits dynamic IP address negotiation, **netconfig** treats each PPP connection as though it were a SLIP line and requires explicit entry of IP addresses for your system and for the target system. You can consider the target system address a starting address with negotiation available when you use PPP.

NOTE During configuration, the **netconfig** utility supplies information to the */etc/hosts*, */etc/strcf*, and */etc/tcp* files. None of these files require additional editing during configuration.

The following procedure describes how to configure PPP between a machine named *emerald* and a machine named *ruby*. After you enter each selected value, press the ⟨Return⟩ key to complete your response:

1. Boot the machine *emerald* in System Maintenance mode by entering the *root* password at the following prompt:

```
Type CONTROL-D to proceed with normal startup (or enter root
passwd for System Maintenance Mode):
```

2. Enter the following command and press ⟨Return⟩:

   netconfig

3. At the Available options menu, enter the number that corresponds to Add a chain. A *chain*, in **netconfig** terms, is a communications configuration that binds networking products together as though by a chain.
4. From the list of available top-level products, enter the number that corresponds to sco_tcp.
5. At the prompt `Select next level of chain to Add or q to quit:`, enter the number that corresponds to ppp0.
6. When a prompt asks you to confirm the selected product chain, enter **y** if the displayed product chain is correct. If the chain is incorrect, enter **n** to return to the **netconfig** main menu. You can then either quit **netconfig** or add a different chain of products.
7. The first time that you configure a PPP connection, follow the **netconfig** prompts to choose a password.
8. At the prompt `Enter the internet address of this interface:`, enter the IP address for your system (*emerald*) in this format: ***nnn.nnn.nnn.nnn.***
9. At the prompt `Enter the netmask for this interface`, enter the netmask in this format: ***nnn.nnn.nnn.nnn.***
10. At the prompt `Enter the internet address of the destination interface:`, enter the IP address for the target system (*ruby*) in this format: ***nnn.nnn.nnn.nnn.***
11. At the prompt `Enter hostname of the destination interface:`, enter the name of the target system; in this example, **ruby**.
12. The prompt `Are these values correct? (y/n)` gives you a chance to change your choices. If you want to alter any of the values, enter **n**, then follow the screen prompts. If the values are correct, enter **y**.
13. At the prompt `Configure gateway?`, enter **n**, unless you plan to establish a PPP/Ethernet or PPP/Token-Ring gateway as described in the SLIP sec-

tion of this chapter. (You only see this prompt if ppp0 is the first driver configured after the loopback driver.)

14. When **netconfig** prompts you to add or remove pseudo ttys, enter **q** if you want to maintain the pseudo ttys currently configured. If you want to change the number of pseudo ttys, enter **y**, and consult the section on administering pseudo ttys in the chapter "TCP/IP network administration" earlier in this document.
15. When you again see the Available options: menu, enter **q**.
16. You see a prompt to relink the kernel. Enter **y** for the first PPP configuration. Subsequent PPP configurations on the same machine do not require relinking the kernel.
17. During kernel relinking, enter **y** to have this kernel boot by default, and **y** again to have the kernel environment rebuilt. This procedure takes a few minutes. When the rebuild is complete, you return to the command line.
18. Reboot *emerald*.
19. Enable the tty line that PPP is using.
20. Make sure that */etc/ppphosts* has the correct entry in the tty column for the chosen connection (modem or direct). The correct entry for a modem connection is a dash "-"; the correct entry for a direct connection is the name of the device that PPP is using (for example, */dev/tty1a*).
21. Use the same procedure to configure the PPP line on *ruby*, making the interface address for *ruby*'s PPP line match the destination address for *emerald*'s PPP line, and making the destination address for *ruby*'s PPP line match the interface address for *emerald*'s PPP line.

If you want to configure more than one PPP line on a machine, repeat the **netconfig** procedure for each line you wish to configure, selecting the option for ppp1, and so on. Make sure that each PPP line is a separate subnet (has a unique network address and hostname).

Adding PPP information to configuration files

The next step in the configuration procedure is to add information to the parameter files used by UUCP and PPP. If your PPP line operates over modems, you might also need to add information to a file that controls the dial activity. Each file has an associated manual page as follows:

File	Description	Manual page	Permissions
/usr/lib/uucp/Systems	remote system info	systems(F)	600
/usr/lib/uucp/Devices	device info	devices(F)	644
/usr/lib/uucp/Dialers	dialing info	dialers(F)	644

You can edit these files directly, using your choice of text editor, or you can run **uuinstall**(ADM), which prompts you for the necessary information.

NOTE If you edit the parameter files directly, create a backup copy of each file and refer to the respective manual page before you make any changes. When you finish editing a file, make sure that the permissions modes are the same as those shown for each file. This is especially important for the *Systems* file, which contains high-security information.

The remainder of this section briefly presents information and examples to help you edit the parameter files either directly or through **uuinstall**. For more details about these files, see the UUCP chapter in your administrator's guide.

Make sure that site names are consistently named in each file.

/usr/lib/uucp/Systems:

```
pppdir Any Direct 2400-9600 - "" \r ogin:--ogin: nppp word: testing
pppmodem Any ACU 9600 5555 "" \r ogin:--ogin: nppp word: testing
```

The first entry is for a PPP direct line called **pppdir** that can operate at speeds from 2400 to 9600 baud. The second entry is for a modem line that uses an Automatic Calling Unit (ACU) that operates at 9600 baud. This dialer is accessed through an internal phone system, such as a PBX, and is reached at extension 5555. Both lines have **testing** as their login keyword.

/usr/lib/uucp/Devices:

```
Direct ttyla - 2400-9600 direct
ACU ttylA - 9600 atdialUSR
```

For this file, the UUCP installation might supply sufficient entries. Make sure that the file provides direct information, as the first entry shows. If PPP is used with a modem, make sure that the file contains a line such as the second entry.

/usr/lib/uucp/Dialers:

This file might already contain the appropriate entries. Refer to the **dialers**(F) manual page and the chapter "Using Modems" later in this guide for instructions on adding information to the *Dialers* file.

Configuring a PPP/Ethernet or PPP/Token-Ring gateway

To set up a machine as a gateway between a PPP network and an Ethernet network, follow the procedure for configuring a SLIP/Ethernet gateway that appears earlier in this chapter. Configuring a gateway machine between a PPP network and a Token-Ring network also uses the same procedure.

Removing PPP

Remove PPP with **netconfig**, and then restore the files that you changed to reflect the PPP line.

1. At the Available options prompt, enter the number corresponding to `Remove a chain`, and press ⟨Return⟩.
2. Follow the directions in the rest of the **netconfig** screen prompts, entering **y** to relink the kernel.
3. If you are removing a PPP/Ethernet or PPP/Token-Ring gateway, remove the network entries from */etc/networks*.
4. If you are removing your last link to another machine, remove the entry for that machine from your *Systems* file for security reasons. You can use **uuinstall** to edit the *Systems* file.

Troubleshooting PPP

To begin troubleshooting PPP, prepare a test environment as follows:

1. Place two computers side-by-side and connect with a direct serial line. Generally, a null modem cable should cross-connect the serial ports between the two computers.
2. Make sure that the PPP software is installed correctly by running **netconfig** and verifying that the display matches your expectations.
3. Be familiar with the "Troubleshooting your system" chapter in the *System Administrator's Guide* supplied with your system.
4. Bring both systems to a known state. For example, reboot both systems, and bring them to multiuser mode.
5. Using multiscreens, monitor the system message log on each system with:

   tail -f /usr/adm/syslog

The following guidelines can help you isolate the problem and find the solution:

Does the serial line function?

For a PPP connection on *tty1a* running at 1200 baud, enter the following command on each machine:

cu -ltty1a -s1200 dir

This command should cause characters typed on one machine to display on the other. If not, make sure that the */usr/lib/uucp* file is correct, and make sure that your cable does not have a short. If the **cu** command does display the

characters on the other machine, then run **netconfig** to make sure that the PPP is installed. If it is installed, then make sure that a device node exists for the tty, and that the serial cable is securely connected.

Does the modem connection work?

For a PPP connection on *tty1a* running at 1200 baud and calling a modem at extension 5555, enter the following command on each machine:

cu -ltty1a -s1200 5555

The modem should dial, connect, and present a login prompt. If you do not see a login prompt after many seconds, type **~%B** to make sure that the remote modem did not accidentally cycle past the correct baud rate. Try **~%B** three times, waiting several seconds in between each try. If you still do not get a login prompt, make sure that the entries in your *Devices* file are correct, and that the remote tty setup is correct.

Next, enter the following command on each machine, specifying the remote hostname:

cu *hostname*

You should get the same results as when you specify the extension number. If you do not, make sure that the entries in your *Systems* are correct.

Does ping work?

Follow the directions in the SLIP troubleshooting section earlier in this chapter to test the connection with the **ping** command. If **ping** does not work, make sure that the hostname listed in */etc/ppphosts* is correct.

Does telnet work?

If you run **telnet** or one of the other utilities for using PPP, and messages do not appear or the command fails, check the */usr/adm/syslog* display. If this does not provide the answer, try disabling and enabling the tty device associated with the PPP serial line. You can also check that the **uutry**(ADM) and **uustat**(C) utilities work. Follow the directions in the Performance and Troubleshooting section of the *System Administrator's Guide*. This chapter has detailed directions for handling error messages such as `DEVICE LOCKED`, `LOGIN FAILED`, or `NO DEVICES AVAILABLE`.

For additional troubleshooting ideas, consult the section on troubleshooting SLIP lines that appears earlier in this chapter.

More PPP information

The following manual pages provide information about PPP:

Manual page	Description
asyhdlc(ADMP)	asynchronous HDLC device driver
ifconfig(ADMN)	configure network parameters
if.ignore(SFF)	specifies which daemons should not be sent over PPP
ppp(ADMN)	PPP login shell
ppp(ADMP)	PPP implementation
pppd(ADMN)	PPP daemon
ppphosts(SFF)	PPP host attributes
strcf(SFF)	STREAMS configuration file format

For more information about how PPP is designed and implemented, refer to RFC 1171 and RFC 1172, both written by Jon Postel and Joyce Reynolds. These documents are available from Jon Postel, RFC Editor, USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292-6695.

Chapter 9

Configuring the BIND name server

With the Berkeley Internet Name Domain (BIND) server, you can create and maintain a distributed host name and address database for computers on your network. Your system is configured by default to use the network hosts file */etc/hosts* found on each computer. If you have a large network, updating every computer's */etc/hosts* file can be time-consuming. Using BIND frees the system administrator from continually updating host name and address data on every machine.

This chapter explains basic BIND concepts, describes BIND configuration tasks, and provides examples of the various BIND configuration files.

The name service

The basic function of the name server is to answer queries from clients about host names and addresses. With the name server, the network is broken into a hierarchy of domains. The name space is organized as a tree, according to organizational or administrative boundaries. Each node, called a domain, is given a label, and the name of the domain is the concatenation of all the labels of the domains from the *root* to the current domain, listed from right to left, separated by dots. A label need only be unique within its domain. The whole space is partitioned into several areas called zones, each starting at a domain and extending down to the leaf domains or to domains where other zones start. Zones usually represent administrative boundaries. An example of a host address for a host at the University of California, Berkeley, would look as follows:

```
monet . Berkeley . EDU
```

The top-level domain for educational organizations is EDU; Berkeley is a subdomain of EDU and monet is the name of the host. Additional top-level domains include:

COM	commercial organizations
GOV	government organizations
MIL	military departments
ORG	miscellaneous organizations

Types of servers

There are several types of servers. These are:

- master servers
- caching-only servers
- remote servers
- slave servers

These types of servers are described in more detail in the following four sections.

Master servers

A "master" server for a domain is the authority for that domain. This server maintains all the data corresponding to its domain. Each domain should have at least two master servers: a primary master, and some secondary masters to provide backup service if the primary is unavailable or overloaded. A server may be a master for multiple domains, being primary for some domains and secondary for others.

Primary

A "primary master" server is a server that loads its data from a file on disk. This server may also delegate authority to other servers in its domain.

Secondary

A "secondary master" server is a server that is delegated authority and receives its data for a domain from a primary master server. At boot time, the secondary server requests all the data for the given zone from the primary master server. This server then periodically checks with the primary server to see if it needs to update its data.

Caching-only servers

All servers are "caching" servers. This means that the server caches the information that it receives for use until the data expires. A caching only server is a server that is not authoritative for any domain. This server services queries and asks other servers that have the authority for the information needed. All servers keep data in their caches until the data expires, based on a time-to-live field attached to the data when it is received from another server.

Remote servers

A "remote" server is an option given to people who would like to use a name server on their workstation or on a machine that has a limited amount of memory and CPU cycles. With this option, you can run all of the networking programs that use the name server without running the name server on the local machine. All of the queries are serviced by a name server that is running on another machine on the network.

Slave server

A "slave" server is a server that always forwards queries it cannot satisfy locally to a fixed list of forwarding servers instead of interacting with the master name servers for the *root* and other domains. The queries to the forwarding servers are recursive queries. There may be one or more forwarding servers, and they are tried in turn until the list is exhausted. A slave and forwarder configuration is typically used when you do not wish all the servers at a given site to be interacting with the rest of the Internet servers.

A typical scenario would involve a number of workstations and a departmental timesharing machine with Internet access. The workstations might be administratively prohibited from having Internet access. To give the workstations the appearance of access to the Internet domain system, the workstations could be slave servers to the timesharing machine, which would forward the queries and interact with other name servers to resolve the query before returning the answer. An added benefit of using the forwarding feature is that the central machine develops a much more complete cache of information that all the workstations can take advantage of. The use of slave mode and forwarding is discussed further under the description of the **named** boot file commands.

Setting up your own domain

When setting up a domain that is going to be on a public network, the site administrator should contact the organization in charge of the network and request the appropriate domain registration form.

NOTE If you ever want to connect to a public network, you should reserve a name before customizing your name server files. Doing so ensures that you do not choose a name that is already in use elsewhere on the network.

An organization that belongs to multiple networks (such as the Internet and BITNET) should register with only one network.

The contacts are as follows:

Internet

Sites that are already on the Internet and need information on setting up a domain should contact HOSTMASTER@NIC.DDN.MIL. You may also want to be placed on the BIND mailing list, which is a mail group for people on the Internet running BIND. This group discusses future design decisions, operational problems, and other related topics. To request placement on this mailing list, send mail to the following address:

```
bind-request@ucbarpa.Berkeley.EDU.
```

If you have access to news over the network, you can also subscribe to the newsgroup *comp.protocols.tcp-ip.domains* for more discussion of domains.

BITNET

If you are on the BITNET and need to set up a domain, contact INFO@BITNIC.

Boot file

The name server uses several files to load its database. The major file used is the boot file. This is the file that is first read when **named** starts up. This tells the server what type of server it is, which zones it has authority over, and where to get its initial data. The default location for this file is */etc/named.boot*. However, this can be changed by setting the **BOOTFILE** variable or by specifying the location on the command line when **named** starts up.

Directory

The directory line specifies the directory in which the name server should run, allowing the other filenames in the boot file to use relative pathnames.

```
directory          /usr/local/lib/named
```

If you have more than a couple of named files to be maintained, you may wish to place the named files in a directory such as */usr/local/domain* and adjust

the directory command properly. The main purposes of this command are to make sure **named** is in the proper directory when trying to include files by relative pathnames with **$INCLUDE** and to allow **named** to run in a location that is reasonable to dump core.

Primary master

The line in the boot file that designates the server as a primary server for a zone looks like the following:

```
primary          Berkeley.Edu    ucbhosts
```

The first field specifies that the server is a primary one for the zone stated in the second field. The third field is the name of the file from which the data is read.

Secondary master

The line for a secondary server is similar to that for the primary, except that it lists addresses of other servers (usually primary servers) from which the zone data is obtained.

```
secondary        Berkeley.Edu    128.32.0.10     128.32.0.4
```

The first field specifies that the server is a secondary master server for the zone stated in the second field. The two network addresses specify the name servers that are primary for the zone. The secondary server gets its data across the network from the listed servers. Each server is tried in the order listed until it successfully receives the data from a listed server. If a filename is present after the list of primary servers, data for the zone is dumped into that file as a backup. When the server is first started, the data are loaded from the backup file if possible, and a primary server is then consulted to check that the zone is still up-to-date.

Caching-only server

You do not need a special line to designate that a server is a caching server. A caching-only server is indicated by the absence of authority lines, such as secondary or primary in the boot file.

All servers should have the following line in the boot file to prime the name server's cache:

```
cache            .          root.cache
```

The period "." specifies the domain. All cache files listed are read in at named boot time and any values still valid are reinstated in the cache and the *root* name server information in the cache files are always used. For information on the cache file, see the section "Initializing the Cache" later in this chapter.

Forwarders

Any server can make use of forwarders. A forwarder is another server capable of processing recursive queries to try to resolve queries on behalf of other systems.

The **forwarders** command specifies forwarders by Internet address as follows:

```
forwarders      128.32.0.10 128.32.0.4
```

There are two main reasons for wanting to do so. First, the other systems may not have full network access and may be prevented from sending any IP packets into the rest of the network and, therefore, must rely on a forwarder that does have access to the full net. The second reason is that the forwarder sees a union of all queries as they pass through the forwarder's server and, therefore, the forwarder builds up a very rich cache of data compared to the cache in a typical workstation name server. In effect, the forwarder becomes a meta-cache that all hosts can benefit from, thereby reducing the total number of queries from that site to the rest of the net.

Slave mode

Slave mode is used if the use of forwarders is the only possible way to resolve queries because of lack of full net access or if you wish to prevent the name server from using other than the listed forwarders. Slave mode is activated by placing the simple command in the boot file:

```
slave
```

If **slave** is used, then you must specify forwarders. When in slave mode, the server forwards each query to each of the forwarders until an answer is found or the list of forwarders is exhausted.

Remote servers

Every remote server on your system requires the file */etc/resolv.conf*. This file designates the name servers on the network that should be queried.

Initializing the cache

The name server needs to know the identities of the authoritative name servers for the *root* domain of the network. To do this, you have to prime the name server's cache with the address of these higher authorities. This is done in a file called *root.cache*. The location of this file is specified in the boot file */etc/named.boot*.

Standard files

There are three standard files used to specify the data for a domain. These files are:

named.local
named.hosts
named.rev.

The *named.local* file specifies the address for the local loopback interface, better known as *localhost,* with the network address 127.0.0.1. The location of this file is specified in the boot file.

The *named.hosts* file contains all the data about the machines in this zone. The location of this file is specified in the boot file.

The *named.rev* file specifies the IN-ADDR.ARPA domain. This is a special domain for allowing address-to-name mapping. Because Internet host addresses do not fall within domain boundaries, this special domain was formed to allow inverse mapping. The IN-ADDR.ARPA domain has four labels preceding it. These labels correspond to the four octets of an Internet address. All four octets must be specified even if an octet is zero. The Internet address 128.32.0.4 is located in the domain 4.0.32.128.IN-ADDR.ARPA. This reversal of the address is awkward to read but allows for the natural grouping of hosts in a network.

Standard resource records

The records in the name server data files are called resource records. The following is a general description of a resource record:

```
{name}    {ttl}    addr-class    Record Type    Record Specific data
```

Resource records have a standard format, as shown above. The first field is always the name of the domain record and it must always start in column 1. For some resource records, the name can be left blank. In such cases, the name of the previous resource record is used. The second field is an optional time-to-live field. This specifies how long this data is stored in the database. When this field is left blank, the default time-to-live is specified in the Start of Authority resource record discussed later in this chapter. The third field is the address class. There are currently two classes: IN for Internet addresses and ANY for all address classes. The fourth field states the type of the resource record. The fields after that are dependent on the type of the resource record. Case is preserved in names and data fields when loaded into the name server. All comparisons and lookups in the name server database are case-insensitive.

The following characters have special meanings:

. A free-standing dot in the name field refers to the current domain.

@ A free-standing "@" in the name field denotes the current origin. You can use origins within configuration files as a shorthand for fully qualified domain names.

.. Two free-standing dots represent the null domain name of the *root* when used in the name field.

\X Here *X* is any character other than a digit (0-9), and \X quotes that character so that its special meaning does not apply. For example, \. can be used to place a dot character in a label.

\DDD Here each *D* is a digit, and \DDD is the octet corresponding to the decimal number described by *DDD*. The resulting octet is assumed to be text and is not checked for special meaning.

() Parentheses are used to group data that crosses a line. In effect, line terminations are not recognized within parentheses.

; A semicolon starts a comment; the remainder of the line is ignored.

* An asterisk signifies a wildcard.

Most resource records have the current origin appended to names if they are not terminated by a ".". This is useful for appending the current domain name to the data, such as machine names, but can cause problems when you do not want this to happen. The following is a good rule of thumb: if the name is not in the domain for which you are creating the data file, end the name with a ".".

Separating data into multiple files

An include line begins with **$INCLUDE** (starting in column 1) and is followed by a filename. This feature is particularly useful for separating different types of data into multiple files. Here is an example:

```
$INCLUDE /usr/named/data/mailboxes
```

The line would be interpreted as a request to load the file */usr/named/data/mailboxes*. The **$INCLUDE** command does not cause data to be loaded into a different zone or tree. This is simply a way to allow data for a given zone to be organized in separate files. For example, mailbox data might be kept separately from host data using this mechanism.

Changing an origin in a data file

Use the **$ORIGIN** command to change the origin in a data file. The line starts in column 1 and is followed by a domain origin. This is useful for putting more than one domain in a data file. For example, */etc/named.hosts* might contain lines of the form:

```
$ORIGIN CC.Berkeley.EDU
[assorted domain data...]
$ORIGIN EE.Berkeley.EDU
[assorted domain data...]
```

The start of authority resource record (SOA)

The Start of Authority record designates the start of a zone. An SOA record includes the following fields:

Name	name of the zone
Origin	name of the host on which this data file resides
Person in charge	mailing address for the person responsible for the name server
Serial number	version number of this data file; this number should be incremented whenever a change is made to the data. (Note that the name server cannot handle numbers over 9999 after the decimal point.)
Refresh	how often, in seconds, a secondary name server is to check with the primary name server to see if an update is needed
Retry	how long, in seconds, a secondary server is to retry after a failure to check for a refresh
Expire	upper time limit, in seconds, that a secondary name server is to use the data before it expires for lack of getting a refresh
Minimum	default number of seconds to be used for the time-to-live field on resource records

There should only be one SOA record per zone.

Here is an example of an SOA record:

```
name {ttl} addr-class SOA     Origin                Person in charge
@          IN         SOA     ucbvax.Berkeley.Edu.  kjd.ucbvax.Berkeley.Edu.(
                      1.1     ; Serial
                      3600    ; Refresh
                      300     ; Retry
                      3600000 ; Expire
                      3600)   ; Minimum
```

The name server resource record (NS)

The name server record (NS) lists a name server responsible for a given domain. The first name field lists the domain that is serviced by the listed name server. There should be one NS record for each master server for the domain. Here is an example of a name server record:

```
{name}  {ttl}   addr-class      NS      Name servers name
@               IN      NS      ucbarpa . Berkeley . Edu.
```

The "@" character specifies the current origin. The address class is IN (Internet addresses), and the record type is name server (NS). The record uses the default ttl (time-to-live) value. Here, the record-specific data is the identity of the name server.

The address resource record (A)

The address record (A) lists the address for a given machine. The name field is the machine name and the address is the network address. There should be one A record for each address of the machine. Here is an example of an address record for a machine named *ucbarpa* with two network addresses:

```
{name}  {ttl}   addr-class      A       address
ucbarpa         IN      A       128.32.0.4
                IN      A       10.0.0.78
```

The host information resource record (HINFO)

The host information resource record (HINFO) is for host-specific data. It lists the hardware and operating system that are running at the listed host. It should be noted that only a single space separates the hardware information and the operating-system information. If you want to include a space in the machine name, you must quote the name. Host information is not specific to any address class, so ANY may be used for the address class. There should be one HINFO record for each host. Here is an example:

```
{name}  {ttl}   addr-class      HINFO   Hardware    OS
                ANY     HINFO   VAX-11/780  UNIX
```

Note that the current release ignores any records that appear after an HINFO record. Thus, you can use only one HINFO record within the file, and it should be the last record in the file.

The well-known services resource record (WKS)

The well-known services record (WKS) describes the well-known services supported by a particular protocol at a specified address. The list of services and port numbers comes from the list of services specified in */etc/services*. There should be only one WKS record per protocol per address. Here is an example of a WKS record:

```
{name}  {ttl} addr-class  WKS   address     protocol  list of services
              IN          WKS   128.32.0.10  UDP       who route timed domain
              IN          WKS   128.32.0.10  TCP       (echo telnet
                                                      discard sunrpc sftp
                                                      uucp-path systat daytime
                                                      netstat qotd nntp
                                                      link chargen ftp
                                                      auth time whois mtp
                                                      pop rje finger smtp
                                                      supdup hostnames
                                                      domain
                                                      nameserver)
```

The canonical name resource record (CNAME)

The canonical name resource record (CNAME) specifies an alias for a canonical name. An alias should be the only record associated with the alias name; all other resource records should be associated with the canonical name and not with the alias. Any resource records that include a domain name as their value (for example, NS or MX) should list the canonical name, not the alias. Here is an example of a CNAME record:

```
aliases {ttl}   addr-class      CNAME   Canonical name
ucbmonet                IN      CNAME   monet
```

The domain name pointer resource record (PTR)

A domain name pointer record (PTR) allows special names to point to some other location in the domain. The following example of a PTR record is used in setting up reverse pointers for the special IN-ADDR.ARPA domain. This line is from the example:

```
named.rev file.
```

In this record, the name field is the network number of the host in reverse order. You only need to specify enough octets to make the name unique. For example, if all hosts are on network 127.174.14, then only the last octet needs

to be specified. If hosts are on networks 128.174.14 and 127.174.23, then the last two octets need to be specified. PTR names should be unique to the zone. Here is an example of a PTR record:

```
name    {ttl}   addr-class      PTR     real name
7.0             IN      PTR     monet.Berkeley.Edu.
```

The mailbox resource record (MB)

The mailbox resource record has a record type of MB. It lists the machine where a user wants to receive mail. The name field is the user's login; the machine field denotes the machine to which mail is to be delivered. Mail box names should be unique to the zone. Here is an example of an MB record:

```
name    {ttl}   addr-class      MB      Machine
miriam          IN      MB      vineyd.DEC.COM.
```

The mail rename resource record (MR)

The mail rename record (MR) lists aliases for a user. The name field lists the alias for the name listed in the fourth field, which should have a corresponding MB record. Here is an example of a mail rename record:

```
name    {ttl}   addr-class      MR      corresponding MB
Postmistress    IN      MR      miriam
```

The mailbox information resource record (MINFO)

The mail information record MINFO creates a mail group for a mailing list. This resource record is usually associated with a mail group, but it can be used with a mail box record. The "name" specifies the name of the mailbox. The "requests" field is where mail such as requests to be added to a mail group should be sent. The "maintainer" is a mailbox that should receive error messages. This is particularly appropriate for mailing lists when errors in members' names should be reported to a person other than the sender. Here is an example of this record:

```
name  {ttl}  addr-class  MINFO  requests      maintainer
BIND         IN          MINFO  BIND-REQUEST  kjd.Berkeley.Edu.
```

The mail group member resource record (MG)

The mail group record (MG) lists members of a mail group.

```
{mail group name}  {ttl}  addr-class  MG  member name
                          IN          MG  Bloom
```

An example for setting up a mailing list is as follows:

```
Bind    IN      MINFO   Bind-Request          kjd.Berkeley.Edu.
        IN      MG      Ralph.Berkeley.Edu.
        IN      MG      Zhou.Berkeley.Edu.
        IN      MG      Painter.Berkeley.Edu.
        IN      MG      Riggle.Berkeley.Edu.
        IN      MG      Terry.pa.Xerox.Com.
```

The mail exchanger resource record (MX)

```
name            {ttl}  addr-class  MX  preference value  mailer exchanger
Munnari.OZ.AU.         IN          MX  0                 Seismo.CSS.GOV.
*.IL.                  IN          MX  0                 RELAY.CS.NET.
```

Mail exchanger records (MX) identify a machine that knows how to deliver mail to a machine that is not directly connected to the network. In the first example above, Seismo.CSS.GOV. is a mail gateway that knows how to deliver mail to Munnari.OZ.AU. but other machines on the network cannot deliver mail directly to Munnari. These two machines may have a private connection or use a different transport medium. The preference value is the order that a mailer should follow when there is more then one way to deliver mail to a single machine. See RFC974 for more detailed information.

Wildcard names containing the character "*" may be used for mail routing with MX records. There are likely to be servers on the network that simply state that any mail to a domain is to be routed through a relay. In the second example above, all mail to hosts in the domain IL is routed through RELAY.CS.NET. This is done by creating a wildcard resource record, which states that *.IL has an MX of RELAY.CS.NET.

Some sample files

The following sections contain sample files for the name server. This covers example boot files for the different types of server and example domain database files.

Caching-only server

```
;
; Boot file for Caching Only Name Server
;
; type  domain  source file or host
;
domain  Berkeley.Edu
cache   .       /etc/named.ca
primary 0.0.127.in-addr.arpa    /etc/named.local
```

Primary master server

```
;
; Boot file for Primary Master Name Server
;
; type  domain  source file or host
;
directory       /usr/local/lib/named
primary Berkeley.Edu    ucbhosts
primary 32.128.in-addr.arpa     ucbhosts.rev
primary 0.0.127.in-addr.arpa    named.local
cache   .       root.cache
```

Secondary master server

```
;
; Boot file for Secondary Name Server
;
; type   domain   source file or host
;
directory /usr/local/lib/named
secondary Berkeley.Edu      128.32.0.4 128.32.0.10 128.32.136.22 ucbhosts.bak
secondary 32.128.in-addr.arpa   128.32.0.4 128.32.0.10 128.32.136.22 ucbhosts.rev.bak
primary  0.0.127.in-addr.arpa       named.local
cache    .       root.cache
```

The /etc/resolv.conf file

```
domain Berkeley.Edu
nameserver 128.32.0.4
nameserver 128.32.0.10
```

root.cache

The addresses shown in this file are for example only, and they do not indicate valid addresses. You need to edit your own *root.cache* file to contain valid addresses.

```
;
; Initial cache data for root domain servers.
;
.       99999999        IN      NS      NS.NIC.DDN.MIL.
        99999999        IN      NS      A.ISI.EDU.
        99999999        IN      NS      AOS.BRL.MIL.
        99999999        IN      NS      C.NYSER.NET.
        99999999        IN      NS      TERP.UMD.EDU.
        99999999        IN      NS      NS.NASA.GOV.
;  Prep the cache (hotwire the addresses).
NS.NIC.DDN.MIL. 99999999        IN      A       192.67.67.53
A.ISI.EDU.      99999999        IN      A       26.3.0.103
A.ISI.EDU.      99999999        IN      A       128.9.0.107
AOS.BRL.MIL.    99999999        IN      A       192.5.25.82
C.NYSER.NET.    99999999        IN      A       192.33.4.12
TERP.UMD.EDU.   99999999        IN      A       128.8.10.90
NS.NASA.GOV.    99999999        IN      A       192.52.195.10
```

named.local

```
@    IN SOA     ucbvax.Berkeley.Edu. kjd.ucbvax.Berkeley.Edu. (
                1       ; Serial
                10800     ; Refresh
                1800      ; Retry
                3600000 ; Expire
                86400 )   ; Minimum
     IN  NS     ucbvax.Berkeley.Edu.
1    IN  PTR    localhost.
```

named.hosts

```
;
;    @(#)ucb-hosts    1.1    (berkeley)    86/02/05
;
@       IN      SOA     ucbvax.Berkeley.Edu. kjd.monet.Berkeley.Edu. (
                        1.1       ; Serial
                        3600      ; Refresh
                        300       ; Retry
                        3600000   ; Expire
                        3600 )    ; Minimum
        IN      NS      ucbarpa.Berkeley.Edu.
        IN      NS      ucbvax.Berkeley.Edu.
localhost       IN      A       127.0.0.1
ucbarpa IN      A       128.32.4
        IN      A       10.0.0.78
        IN      HINFO   VAX-11/780 UNIX
arpa    IN      CNAME   ucbarpa
ernie   IN      A       128.32.6
        IN      HINFO   VAX-11/780 UNIX
ucbernie        IN      CNAME   ernie
monet   IN      A       128.32.7
        IN      A       128.32.130.6
        IN      HINFO   VAX-11/750 UNIX
ucbmonet        IN      CNAME   monet
ucbvax  IN      A       10.2.0.78
        IN      A       128.32.10
        IN      HINFO   VAX-11/750 UNIX
        IN      WKS     128.32.0.10 UDP syslog route timed domain
        IN      WKS     128.32.0.10 TCP ( echo telnet
                        discard sunrpc sftp uucp-path systat
                        daytime netstat qotd nntp link chargen
                        ftp auth time whois mtp pop rje finger
                        smtp supdup hostnames domain nameserver )
vax     IN      CNAME   ucbvax
toybox  IN      A       128.32.131.119
        IN      HINFO   Pro350 RT11
toybox  IN      MX      0  monet.Berkeley.Edu
miriam  IN      MB      vineyd.DEC.COM.
postmistress    IN      MR      Miriam
Bind    IN      MINFO   Bind-Request kjd . Berkeley . Edu .
        IN      MG      Ralph . Berkeley . Edu .
        IN      MG      Zhou . Berkeley . Edu .
        IN      MG      Painter . Berkeley . Edu .
        IN      MG      Riggle . Berkeley . Edu .
        IN      MG      Terry . pa . Xerox . Com .
```

named.rev

```
;
;    @(#)ucb-named.rev    1.1    (Berkeley)    86/02/05
;
@       IN      SOA     ucbvax.Berkeley.Edu. kjd.monet.Berkeley.Edu. (
                        1.1     ; Serial
                        10800   ; Refresh
                        1800    ; Retry
                        3600000 ; Expire
                        86400 ) ; Minimum
        IN      NS      ucbarpa.Berkeley.Edu.
        IN      NS      ucbvax.Berkeley.Edu.
4.0     IN      PTR     ucbarpa.Berkeley.Edu.
6.0     IN      PTR     ernie.Berkeley.Edu.
7.0     IN      PTR     monet.Berkeley.Edu.
10.0    IN      PTR     ucbvax.Berkeley.Edu.
6.130   IN      PTR     monet.Berkeley.Edu.
```

Additional sample files

The following sections contain an additional set of sample files for the name server. See the manual page entry for **named**(ADMN) for more information on specific entries in these files.

named.boot

```
;
; Name Server boot file for Domain mynet.COM
;
; Type  Domain  Source file or Host
;
domain  mynet.COM
primary mynet.COM       /etc/named.data/mynet-hosts
cache   .       /etc/named.data/root.cache
primary 200.9.192.in-addr.arpa  /etc/named.data/mynet-host.s.rev
primary 0.0.127.in-addr.arpa    /etc/named.data/named.local
```

root.cache

```
;
; Initial cached data for root domain servers.
;
;.                      99999999 IN NS  USC-ISIB.ARPA.
;                       99999999 IN NS  BRL-AOS.ARPA.
;                       99999999 IN NS  SRI-NIC.ARPA.
;
; Insert your own name servers here
;
;.                      99999999 IN NS  tandy.mynet.COM
;
; Prep the cache (hotwire the addresses)
;
tandy.mynet.COM.                99999999 IN A   192.9.200.2
;
; Root servers go here
;
tandy.mynet.COM.                99999999 IN A   192.9.200.2
;SRI-NIC.ARPA.          99999999 IN A   10.0.0.51
;USC-ISIB.ARPA.         99999999 IN A   10.3.0.52
;BRL-AOS.ARPA.          99999999 IN A   128.20.1.2
;BRL-AOS.ARPA.          99999999 IN A   192.5.22.82
```

named.local

```
;
; Don't forget to increment the serial number in
; named.soa
;

$INCLUDE /usr/lib/named/mynet.soa
1       IN PTR  localhost.
```

mynet-host.s.rev

```
;
; Don't forget to increment the serial number in
; named.soa
;

$INCLUDE /usr/lib/named/mynet.soa

1       IN      PTR     merlin
2       IN      PTR     tandy
3       IN      PTR     tvi
```

mynet.soa

```
;
; Don't forget to increment the serial number when you
; change this.  SCCS or RCS might be a good idea here.
;

@.              IN SOA  tandy.mynet.COM.   root.tandy.mynet.COM. (
                        1.0     ; Serial
                        3600    ; Refresh
                        300     ; Retry
                        3600000 ; Expire
                        3600 )  ; Minimum
                IN NS   tandy.mynet.COM.
```

Domain management

This section contains information for starting, controlling, and debugging **named**(ADMN), the Internet domain name server.

Starting the name server

The host name should be set to the full domain style name (that is, *monet.Berkeley.EDU.*) using **hostname**(TC). The name server is started automatically if the configuration file */etc/named.boot* is present. Do not attempt to run **named** from **inetd**(ADMN). This continuously restarts the name server and defeats the purpose of having a cache.

/etc/named.pid

When **named** is successfully started, it writes its processID into the file */etc/named.pid*. This is useful to programs that want to send signals to **named**.

/etc/hosts

The **gethostbyname** library call can detect whether **named** is running. If it is determined that **named** is not running, it looks in */etc/hosts* to resolve an address. This option was added to allow **ifconfig**(ADMN) to configure the machine's local interfaces and to enable a system manager to access the network while the system is in single-user mode. It is advisable to put the local machine's interface addresses and a couple of machine names and addresses in */etc/hosts*, so the system manager can copy files from another machine with **rcp** when the system is in single-user mode. The format of */etc/hosts* has not changed. See *hosts*(SFF) for more information. Because the process of reading */etc/hosts* is slow, it is not advisable to use this option when the system is in multiuser mode. Note that all network software must be manually started in single-user mode.

Reload

There are several signals that can be sent to the **named** process to have it do tasks without restarting the process. The **SIGHUP** signal causes **named** to read *named.boot* and reload the database. All previously cached data is lost. This is useful when you have made a change to a data file and you want **named**'s internal database to reflect the change.

Debugging

When **named** is running incorrectly, look first in */usr/adm/syslog* and check for any messages logged by **syslog**. Next, send it a signal to see what is happening.

SIGINT dumps the current database and cache to */usr/tmp/named_dump.db*. This should give you an indication as to whether the database was loaded correctly.

NOTE **named** is built with DEBUG defined. The following signals work when DEBUG is defined.

SIGUSR1 - turns on debugging. Each signal following USR1 increments the debug level. The output goes to */usr/tmp/named.run*.

SIGUSR2 - turns off debugging completely

For more information on other signals and debugging BIND, refer to the **named** manual page.

More BIND information

The following Requests for Comments (RFC's) contain additional information about BIND:

RFC	Title and Information
RFC819	The Domain Naming Convention for Internet User Applications, Su, Z., Postel, J., 8/82.
RFC882	Domain Names - Concept and Facilities, Mockapetris, P., 11/83.
RFC883	Domain Names - Implementation and Specification, Mockapetris, P., 11/83.
RFC973	Domain System Changes and Observations, Mockapetris, P., 2/86.
RFC974	Mail Routing and The Domain System, Partridge, C., 2/86.

These RFC's are available from the Network Information Center, Suite 200, 14200 Park Meadow Dr., Chantilly, VA 22021

Chapter 10
Gateways and routing

To exchange data between hosts that are attached to different networks, certain hosts, called "gateways", are responsible for exchanging routing information and forwarding data from one network to another until the data reaches its final destination. Gateways can use several different protocols to exchange reachability and routing information among themselves. Some of these protocols are: EGP (Extended Gateway Protocol), BGP (Border Gateway Protocol), RIP (Routing Information Protocol), and the HELLO protocol.

In large interconnected networks that are connected to the Internet, each portion of the network that is under a separate local administration is called an Autonomous System (AS) and is assigned a unique number by the DDN Network Information Center (NIC). Each AS is connected to a common core network via an exterior gateway. All the exterior gateways exchange host routing or reachability information among themselves using an exterior gateway protocol, such as EGP or BGP.

Within an AS there may be a single Local Area Network (LAN) or many interconnected networks that are linked via interior gateways. These gateways use an interior gateway protocol, such as RIP or HELLO, to exchange routing information.

This chapter describes two programs that are designed to exchange routing information and that you can run on a host to have it function as a gateway: the **gated**(ADMN) daemon and the **routed**(ADMN) daemon. The **gated** daemon allows a host to function as both an exterior and interior gateway, whereas the **routed** is intended only for interior gateways. Only one of these daemons can run on your host at any one time.

Table 10-1 summarizes the differences between the two:

Table 10-1 routed/gated comparison

Feature	routed	gated
Protocol used	RIP	EGP, BGP, RIP, HELLO
Autonomous system use	within an AS	within and between ASs
Route restricting	none	ASs can be excluded

Running routed

If you wish to run **routed**, ensure that the file */etc/gated.conf* does not exist. If it does, you must remove it, move it to another directory, or rename it. Otherwise, the **gated** daemon will execute instead of **routed** the next time your host reboots. The **routed** daemon uses the Routing Information Protocol (RIP) to exchange routing information and the */etc/gateways* file to initialize static routes to distant networks. The */etc/gateways* file is described in the **routed** manual page.

When **routed** starts, it reads the */etc/gateways* file (if it exists), and installs the routes defined there into its routing table. It then broadcasts on each local network to find other hosts running **routed**. If such hosts exist, the routing daemons cooperate in maintaining a globally consistent view of routing within an Autonomous System. This view can be extended to include remote networks also running **routed** by setting up suitable entries in */etc/gateways*.

Another approach to routing to non-local networks is to define a default or wildcard route to a smart gateway and depend on the gateway to provide ICMP routing-redirect information to dynamically create a routing database. This is done by adding an entry to */etc/tcp* that invokes the **route**(ADMN) command, as in the following example:

```
/etc/route add default smart-gateway 1
```

The **route** command is used to manually manipulate routing tables. See the **route** manual page for more information.

The host uses the default route as a last resort in routing packets to their destinations. Assuming the gateway to which packets are directed can be reached to generate the proper ICMP routing-redirect messages, the host adds routing table entries based on the information supplied. This approach has certain advantages over using the **routed** daemon, but it is unsuitable in an environment where there are only bridges (for example, bridges do not generate ICMP routing-redirect messages.) Further, if the smart gateway goes down, there is no way to maintain service except to alter the routing table manually using the **route** command.

The host always listens to, and processes, ICMP routing-redirect messages, and so it is possible to combine both of the above facilities. For example, the **routed** daemon might be used to maintain up-to-date information about routes to local networks, while employing the wildcard routing techniques for distant networks.

Running gated

Before you can run **gated** as your gateway daemon, you must first create the configuration file */etc/gated.conf*. Before creating this file, you should be able to answer the following questions:

1. What is your Automonous System ID (AS ID)?
2. What protocol(s) will you use?
3. With which exterior and interior gateways will you be exchanging routing or reachability information?
4. What filtering controls are required for this gateway? For example, which AS's do you want to exclude?

Next, examine the information provided for each protocol that you wish to support from the appropriate protocol configuration files provided with your TCP/IP software distribution. The four protocols and related configuration files are:

Protocol	Filename
EGP	/etc/gated.egp
BGP	/etc/gated.bgp
RIP	/etc/gated.rip
HELLO	/etc/gated.hello

Read the *gated.conf*(SFF) manual page carefully while examining these files. Using the provided files as a model, copy the appropriate statements into the *gated.conf* file and tailor them to your site requirements. Note that comments in the configuration file begin with a number sign "#" and statements are terminated with a semicolon ";". Also, the definition, protocol, route, and control statements must be specified in the order just listed.

NOTE If the */etc/gated.conf* file does not exist, **routed** is started as the gateway daemon instead of **gated** the next time your host reboots.

When first running **gated**, you should execute it with the **-n** option. This ensures that **gated** does not modify the kernel routing table. When you have finished testing the gateway and are satisfied that it is working correctly, restart the daemon without the **-n** option.

The basic flow of execution of **gated** when reading the configuration file is as follows:

1. Enable **traceoptions**.
2. If BGP or EGP is being used as the gateway protocol, establish the Autonomous System ID.
3. Enable or disable the appropriate protocols.
4. Specify the protocol attributes.

Sample configuration file

Below is a sample *gated.conf* configuration file for a gateway connected to the Internet that has an IP address of 128.212.63.2 and that is part of an Autonomous System (AS) assigned an AS ID of 519. This gateway uses RIP to exchange routing information within the AS. This gateway also uses BGP to exchange routing information with another gateway on the Internet (the "peer gateway") that has an IP address of 192.35.52.90 and that is part of an Autonomous System assigned an AS ID of 963.

```
traceoptions internal external route bgp ;
autonomoussystem 519 ;

rip yes {
 interface wdn0 ;
 trustedgateways 128.212.64.1 ;
} ;

bgp yes {
 peer 192.35.52.90 linktype horizontal asin 963 interface e3B0 ;
} ;

propagate proto bgp as 963 metric 10 {
 proto rip {
  announce all;
 };
};

propagate proto rip metric 2 {
 proto bgp as 519 {
  announce all;
 };
};
```

Following is a detailed explanation of all of the statements in the configuration file.

```
traceoptions internal external route bgp ;
```

This statement turns on tracing for internal errors, external errors and information messages. It also records changes to the **gated** routing table and all BGP packets that are sent and received.

```
autonomoussystem 519 ;
```

This statement indicates that the gateway is a part of the Autonomous System whose AS ID is 519.

```
rip yes {
 interface wdn0 ;
 trustedgateways 128.212.64.1 ;
} ;
```

This statement says that the Routing Information Protocol (RIP) is to be used to exchange routing information within our Autonomous System. The hardware interface to our Autonomous System is controlled by the SCO LLI driver *wdn0*. Finally, this statement says that we will update our RIP routing tables in response to new routing information only from the gateway with IP address 128.212.64.1.

```
bgp yes {
 peer 192.35.52.90 linktype horizontal asin 963 interface e3B0 ;
} ;
```

This statement says to use the Border Gateway Protocol (BGP) to exchange routing information with the peer gateway at IP address 192.35.52.90. The peer gateway is part of another Autonomous System (**linktype horizontal**) that has an AS ID of 963. Finally, our connection to the gateway is through the LLI driver *e3B0*.

```
propagate proto bgp as 963 metric 10 {
 proto rip {
  announce all;
 };
};
```

This statement says to advertise all of our RIP routes to the peer gateway using a metric of 10. This is necessary because BGP currently lacks a specification for advertising a default route to a peer gateway, so we must announce them all. Note that a metric must be specified to prevent the default metric from being used, because the default value indicates that the Autonomous System is unreachable.

NOTE The **gated** daemon uses the metric signifying "unreachable" as the default metric value for all supported protocols. Consequently, you should always specify a metric in the protocol statements that you place in the configuration file.

```
propagate proto rip metric 2 {
 proto bgp as 519 {
  announce all;
 };
};
```

This statement says to advertise all of our BGP routes to hosts within the Autonomous System using a metric of 2. Announcing all routes is useful in Autonomous Systems that have more than one external gateway.

Assuming the peer gateway uses the same interface cards and also uses RIP to advertise routes within its Autonomous System, its configuration file can be identical to the one just described, except for the following statements:

```
autonomoussystem 519 ;

bgp yes {
 peer 128.212.0.3 linktype horizontal asin 519 interface e3B0 ;
} ;

propagate proto rip metric 2 {
 proto bgp as 519 {
  announce all;
 };
};
```

Note that the Autonomous System ID and the IP address have been changed.

More gated information

To get more information about gateway daemons, configuration files and commands, consult the following manual pages:

Manual page	Information provided
gated(ADMN)	gated daemon
gated.conf(SFF)	gated configuration file
routed(ADMN)	routed daemon
route(ADMN)	manually modifying routing tables
ripquery(ADMN)	querying RIP gateways

To get more information about gateway protocols, consult the following Request for Comments (RFCs).

Protocol	RFC
EGP	911, 904
BGP	1163, 1164
RIP	1058
HELLO	891

To obtain an Autonomous System ID for your network or collection of networks, or to obtain copies of RFC's, contact the Network Information Center, Suite 200, 14200 Park Meadow Drive, Chantilly, VA 22021

Chapter 11

Configuring and using SNMP

The Simple Network Management Protocol (SNMP) monitors and controls TCP/IP-based networks. SNMP allows the retrieval and alteration of networking information maintained by hosts and routers attached to a network. A network administrator can use SNMP to diagnose and correct network problems from remote hosts.

By using the SCO SNMP implementation, a network administrator can gather information such as routing entries, interface status, and protocol statistics. If problems are encountered, the administrator can manipulate items such as the ARP cache and the routing table to add, delete, and modify incorrect entries.

This chapter describes SNMP commands and files, explains how to configure SNMP, and shows some examples of how you might use SNMP to troubleshoot your network.

Basic concepts

There are three basic components of SNMP: the protocol itself, described in detail in RFC 1157; SMI, the Structure of Management Information (RFC 1155); and MIB, the Management Information Base (RFC's 1156 and 1213). For information on obtaining these RFC's (Requests for Comments), see the section "More SNMP information" later in this chapter.

The SNMP protocol

SNMP is a simple protocol with four basic operations: **get, get-next, set,** and **trap.** Each operation is encoded in a separate Protocol Data Unit (PDU). An additional PDU is defined for replying to **get, get next,** and **set** operations. The PDUs are carried through the network in the User Datagram Protocol (UDP) via UDP ports 161 and 162.

Although using an unreliable protocol such as UDP may seem non-intuitive at first, it has definite advantages. For example, sending UDP packets through a network that is damaging or corrupting packets is faster than establishing a TCP connection. This is because SNMP requires only one packet to get through for the request, and an additional one for the response. All TCP implementations would require three packets just to establish a connection.

SMI: Structure of Management Information

The Structure of Management Information (SMI), described in detail in RFC 1155, is a framework that describes the basic types of information that can be manipulated by SNMP. It provides a skeleton that specifies the basic format and hierarchy of management data, but does not describe the objects that can be managed. Instead, it describes the building blocks from which the managed objects are constructed.

A fundamental concept of the SNMP is the notion of *object identifiers.* An object identifier (OID) is a tag that allows a management entity to refer unequivocally to a particular object. Object identifiers are allocated in a tree fashion. The value of the object identifier is a series of integers that refer to a particular traversal of the object tree. Figure 11-1 shows the object identifier hierarchy.

The root of the OID tree has no label. Currently, there are three children of the root, CCITT(0), ISO(1), and Joint-ISO-CCITT(2). The ISO node has many children, one of which is org(3), which is allocated for international organizations. Under org(3) is the U.S. Department of Defense, dod(6), which has the child internet(1).

The name *{ iso org dod internet }* is a symbolic representation for the integer series *1.3.6.1.* Both refer to the object identifier of the Internet sub-tree. In practice, *1.3.6.1* can simply be referred to as "Internet". *{ iso org dod internet }, 1.3.6.1,* and *internet* are all different ways of identifying the same object. In SNMP PDUs, only the numeric sequences are used.

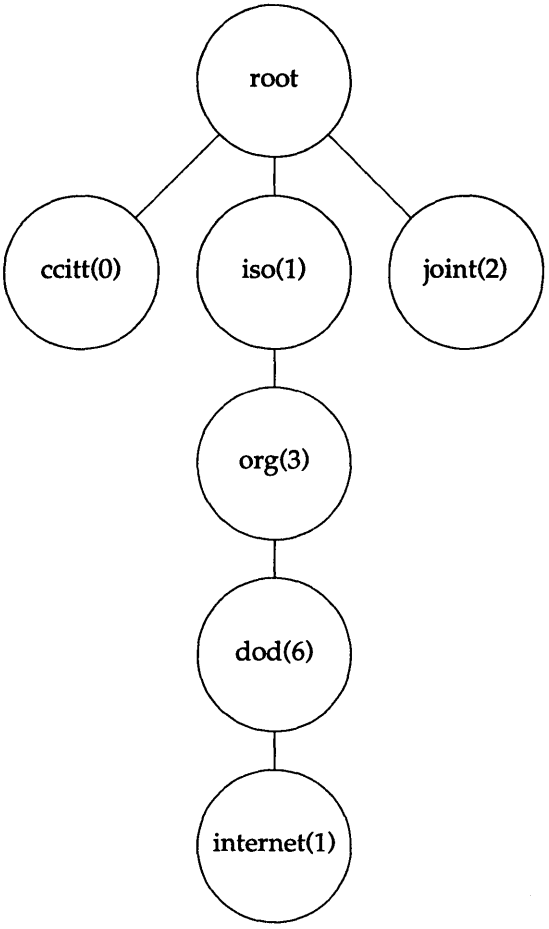


Figure 11-1 Object identifier hierarchy

To assure that object identifiers are unique, each organization is responsible for a particular section of the OID tree. Just as ISO and CCITT have responsibility for their portions, the Internet Activities Board (IAB) has responsibility for the Internet portion.

To allow vendors to support objects that may not be defined in the standard MIB, the IAB reserves a portion of the OID tree for **enterprises** MIBs. This provides vendors with the flexibility they need.

The SMI defines basic data types that make it convenient to describe managed objects. These types are: **ipAddress, Counter, Gauge, TimeTicks, NetworkAddress,** and **Opaque**. Readers who are interested in the precise semantics of these objects are encouraged to read the RFC for further information.

MIB: the Management Information Base

The Management Information Base (MIB), defined in RFC 1156 (MIB-I) and RFC 1213 (MIB-II), lists the standard objects that any SNMP implementation should support. MIB-II provides a superset of those objects found in MIB-I.

Each object in the MIB has two important characteristics. The first is its Object ID (OID). The second is its type. Object types are constructed from the fundamental types defined in the SMI.

The MIB is divided into logically related groups of objects. The important ones are: **system, interfaces, tcp, udp, ip, egp, icmp,** and **snmp.**

The **system** group contains general information about the network node. An example of this would be the physical location of the node, for example, "R&D Facility, 3rd floor machine room."

The **interfaces** group contains information about network interfaces, such as Ethernet® and point-to-point links. Information is kept about such items as interface status (for example, up or down), packet counts, and so on.

The rest of the groups contain information about the particular protocol to which they refer. This includes items such as the number of packets received with a particular protocol type, and number of packets received with incorrect checksums.

Other concepts

In addition to the basic concepts described in the previous section, you should be familiar with the concepts described in the following sections.

Agents and management stations

Systems using SNMP are divided into two categories: **management stations** (sometimes called clients) and **agents** (sometimes called servers). The management station is the system that issues a query; the agent is the system that is being queried.

Although there is no architectural reason that a system cannot have both types of functionality, the bulk of SNMP agents are usually found in dedicated routers and other specialized network equipment that is not capable of acting as a management station.

Using the SCO SNMP implementation, time-sharing systems and workstations can act as both agent and management stations.

Traps

Although SNMP is normally used in a synchronous manner, in which systems poll each other periodically, it does include a mechanism for asynchronous events. In SNMP, these are referred to as "traps." Traps are sent by agents to management stations when an abnormal event, such as a cold reboot, occurs.

Authentication

In the current implementation of SNMP, authentication is based on a field in the PDU called the **community name**. When an agent receives a request, it validates the community name/IP address pair of the sender by comparing it against a specified list (found in */etc/snmpd.comm*) of acceptable management stations. If the sender is not listed, or does not have the appropriate access permissions, the packet is discarded. Optionally, an **authentication** trap may be sent to a specified group of management stations, informing them that an invalid access has been attempted.

Overview of the SCO implementation

As stated earlier, the SCO SNMP product provides the functionality of both agent and management station. A system running the software can monitor other systems, and be monitored as well.

NOTE The ability to operate as a true management station with full graphics capabilities is provided by third-party software that operates over SCO SNMP . Contact your software vendor for more information.

The SNMP Agent is implemented as a UNIX system daemon. There are three configuration files associated with the daemon. These are: */etc/snmpd.conf*, */etc/snmpd.comm*, and */etc/snmpd.trap*. The agent is started when the system enters the multiuser state (traditionally run-level 2). Upon startup, the agent reads configuration information from its configuration files, then begins listening for SNMP requests on the SNMP port. If configured to send traps, the agent also notifies the appropriate management stations that it is up. This is done by sending a "cold-start" trap to the systems listed in */etc/snmpd.trap*.

Example 11-1 shows a sample of that file:

Example 11-1 Sample snmpd.trap file

```
#
# community address       port
#
public      128.212.64.90 162
private     128.212.65.12 162
```

/etc/snmpd.conf contains the basic information that the agent needs as part of the **system** group. These are the types of SNMP and TCP/IP software, the object identifier of the agent, the name of the person responsible for the system, and the location of the system. Generally, only the contact field and the location field require changes. Example 11-2 shows a sample of */etc/snmpd.conf*:

Example 11-2 Sample snmpd.conf file

```
#
descr=SNMPD Version 3.0 for SCO UNIX
objid=SCO.1.0
contact=Trevor Jones x 256
location=First Floor Computer Room
```

/etc/snmpd.comm contains a list of community/IP address pairs from whom the agent accepts queries. Along with each pair is an access field that controls whether access is allowed, and if so, whether the access granted is read-only or read-write. In addition, the special community ***public***, with an address of ***0.0.0.0***, restricts or allows access to all other members of the public community.

By default, */etc/snmpd.comm* contains one entry: `public 0.0.0.0 read`. This allows read access to all members of the public community.

Example 11-3 shows a sample of */etc/snmpd.comm*:

Example 11-3 Sample snmpd.comm file

```
# community address       access
#
public      0.0.0.0       none
private     128.212.64.90 read
ops         128.212.64.2  write
```

In addition to the agent, there are user commands that provide management station functionality to the system administrator. These are: **getid**(ADMN), **getmany**(ADMN), **getnext**(ADMN), **getone**(ADMN), **getroute**(ADMN), **setany**(ADMN), **snmpstat**(ADMN), and **trap_rece**(ADMN).

The **snmpstat**(ADMN) command provides the administrator with an easy way of retrieving information such as routing tables, address translation tables, interface status, and so on. Using **snmpstat**, a network administrator can quickly detect error conditions on a misbehaving network node.

The remaining commands provide the administrator with the ability to retrieve or modify arbitrary objects. Many vendors have their own **enterprises** MIBs in addition to the standard MIB. Because these continually evolve, it may well be the case that a particular node maintains information above that which is conveniently accessed via **snmpstat**. The additional commands can provide access to these **enterprises** MIBs.

Configuring the SNMP agent

SNMP configuration consists of running a special configuration script, **mkdev snmp**. Your SCO TCP/IP Runtime System software, including the TCPRT and SNMPRT packages, must be fully installed and configured before you run this command. To configure SNMP:

1. Log in to the system as *root*.
2. Enter the following command at your prompt and press ⟨Return⟩:

   mkdev snmp

   NOTE If you did not install the complete TCPRT and SNMPRT packages, an error message appears and you return to your operating system prompt. Install the missing packages and run **mkdev snmp** again.

3. The following message appears:

   ```
   Do you wish to install or delete the SNMP agent (i/d/q)
   ```

   Type **i** to install the agent.
4. A screen appears describing the configuration process. After you read it, type **y** to continue the configuration process. Typing **n** or **q** exits the configuration script.
5. A screen appears describing two configurable system group entries found in */etc/snmpd.conf*, sysContact (the name of a person responsible for the system) and sysLocation (the physical location of the system). Accept the default values by pressing ⟨Return⟩, or enter a value of your own and press ⟨Return⟩ when done.

6. A screen appears describing the Community Setup. Information you enter here is placed in the file */etc/snmpd.comm*, and it is read when your system receives requests from other computers.
   - Enter a community name and press ⟨Return⟩.
   - Enter the IP address of a host within that community and press ⟨Return⟩.
   - Enter the permissions for that host (READ, WRITE, or NONE), and press ⟨Return⟩.
7. If you have more hosts to configure, repeat the previous step, entering the community name of the next host. After you have entered and configured all hosts, type **quit** at the community prompt.
8. A screen appears describing the Trap Systems Setup. Information you enter here is placed in the file */etc/snmpd.trap*, which contains a list of management stations to be notified when an abnormal event occurs on your system.
   - Enter a community name and press ⟨Return⟩.
   - Enter the IP address of a host within that community and press ⟨Return⟩.
9. If you have more hosts to configure, repeat the previous step, entering a community and its host IP address. After you have entered and configured all hosts, type **quit** at the community prompt.

   The configuration is complete, and you return to your operating system prompt.
10. You can start the **snmpd** daemon now by entering **snmp start** at your prompt, or you can wait until the next time you bring the system into multiuser mode, when the daemon starts automatically.

The configuration script described above handles most configuration needs. However, you may want to edit the files by hand at a later time.

Full documentation on the SNMP agent and its configuration files can be found in the **snmpd**(ADMN), *snmpd.comm*(SFF), *snmpd.conf*(SFF), and *snmpd.trap*(SFF) manual page entries.

Using the SNMP commands

This section gives a brief overview of using some of the commands provided in the release. Before doing that, it is useful to explain some of the MIB variables used in the examples.

The following variables are from the **system** group: **sysDescr**, **sysContact**, and **sysName**. These are the description of the node, the name of the person responsible for the node, and the name of the node.

The following variables are from the **interfaces** group: **ifDescr, ifOperStatus, ifType,** and **ifPhysAddress** variables. These are the name, status, type, and physical address associated with a particular interface.

The following variables are from the **ip** group, and cover routing table information: **ipRouteDest, ipRouteNextHop,** and **ipRouteType.** These are the destination, next hop of the route entry, and the type of route (for example, direct or indirect).

The easiest command to use for retrieving specific variables is **getone**(ADMN). The syntax of **getone** is **getone node community variable.** So, in order to ask node *grinch* for its system description, one would use the command **getone grinch public sysDescr.0.** This assumes that **grinch** accepts requests with a community of **public.** Example 11-4 shows two examples of the use of **getone.**

Example 11-4 getone example

```
# getone grinch public sysDescr.0

Name: sysDescr.0
Value: SNMPD Version 3.0 for SCO UNIX

# getone grinch public sysName.0

Name: sysName.0
Value: grinch.i88.isc.com
```

The **getid**(ADMN) command retrieves a subset of the **system** group from a node. The syntax is **getid node community.** Example 11-5 shows an example of **getid.**

Example 11-5 getid example

```
# getid grinch public

Name: sysDescr.0
Value: SNMPD Version 3.0 for SCO UNIX

Name: sysObjectID.0
Value: SCO.1.0

Name: sysUpTime.0
Value: 178892
```

The **sysUpTime** variable is the number of clock ticks since the agent was restarted; each tick is 1/100th of a second.

The **getmany**(ADMN) command is useful for retrieving many variables at a time. The syntax of **getmany** is **getmany node community variable.** For example, the entire **system** group could be retrieved using the command

getmany node community system. Or, information about all interfaces attached to the node could be retrieved with the command **getmany node community interfaces.** Example 11-6 shows an example of using **getmany** to get the **system** group.

Example 11-6 getmany example

```
# getmany grinch public system

Name: sysDescr.0
Value: SNMPD Version 3.0 for SCO UNIX

Name: sysObjectID.0
Value: SCO.1.0

Name: sysUpTime.0
Value: 206730

Name: sysContact.0
Value: Trevor Jones x256

Name: sysName.0
Value: grinch.i88.isc.com

Name: sysLocation.0
Value: First Floor Computer Room

Name: sysServices.0
Value: 72
```

The **snmpstat**(ADMN) command provides a subset of the functionality of **getone**, and **getmany**, but has a much friendlier user interface. The **snmpstat** command can display routing, address translation, system, interface, and TCP connection information in a form that is easier to understand. The syntax of **snmpstat** is **snmpstat [options] [node] [community].** If you specify a **community**, you must also specify a node name. If no **community** argument is specified, the default community of **public** is used. With no options specified, **snmpstat** displays the table of transport (TCP and UDP) endpoints. Table 11-1 lists the options available to **snmpstat.**

Table 11-1 snmpstat options

Option	Description
-S	show the **snmp** group
-a	show the address translation table
-i	show statistics from the **interface** group
-n	print addresses numerically for active endpoints
-r	show the routing table
-s	show the **system** group
-t	show all connections

Example 11-7 shows an example of using **snmpstat** to retrieve the **system** group.

Example 11-7 snmpstat system group example

```
# snmpstat -s grinch public
System Group
Description: SNMPD Version 3.0 for SCO UNIX
ObjectID: SCO.1.0
UpTime: 1 hour, 1 minute, 40 seconds, 95 hundredths
Contact: Trevor Jones x256
Name: grinch.i88.isc.com
Location: First Floor Computer Room
Services: applications, end-to-end
```

snmpstat takes care of formatting the information in a way that is easier to understand. For example, **snmpstat** converts the raw timeticks of the agent uptime into a human-readable time.

Example 11-8 shows an example of using **snmpstat** to retrieve routing information.

Example 11-8 snmpstat routing example

```
# snmpstat -r grinch public
Routing table
Destination          Gateway        Metric Type   Proto   Interface
0.0.0.0              houdini        3      rem    rip     ni0
localhost            localhost      0      dir    local   lo0
isc-i88-backbone     grinch         0      dir    local   ni0
isc-i88-dev-net      mcfeely        2      rem    rip     ni0
```

In this example, four routes are returned from *grinch*. Two of them are directly attached. These are the routes for *localhost* and the *isc-i88-backbone* subnet. Two others have been acquired by the RIP routing protocol from the hosts *houdini* and *mcfeely*.

The **setany**(ADMN) command can modify variables. Its basic syntax is **setany node community variable type value**. The legal values for ***type*** are given in Table 11-2. For a complete explanation of these types, refer to RFC 1155.

Table 11-2 setany variable types

Object type	setany argument
Integer	-i
Octet string	-o
Object identifier	-d
IP address	-a
Counter	-c
Gauge	-g
Time_ticks	-t
Display string	-s
Null	-n

Example 11-9 shows an example of using **setany** to alter the status of an interface. The status is verified before and after the change to ensure that the change took place correctly. Changing the status from 1 to 2 marks the interface down.

Example 11-9 Altering interface status

```
# getone grinch public ifAdminStatus.2
Name: ifAdminStatus.2
Value: 1

# setany grinch public ifAdminStatus.2 -i 2
Name: ifAdminStatus.2
Value: 2

# getone grinch public ifAdminStatus.2
Name: ifAdminStatus.2
Value: 2
```

Using SNMP to correct problems

In this section, several "real world" examples are discussed to demonstrate the capabilities of SNMP.

Obtaining remote system contacts

Suppose that the administrator notices a number of local connection attempts to a restricted service (such as NNTP, the USENET News Transfer Protocol,) from an unknown host. The administrator can use **snmpstat** to determine some information about the host if the unknown host is running SNMP. Example 11-10 shows a possible scenario.

Example 11-10 Verifying the origin of a network connection

```
# snmpstat grinch public
Active Internet connections
Proto  Local Address     Foreign Address       (state)
tcp    grinch.nntp       bach.CS.ESU.EDU.1021 ESTABLISHED

# snmpstat -s bach.CS.ESU.EDU
System Group
Description: SNMPD VERSION 9.4.0.0 SUN 3/260 - SUNOS3.5
ObjectID: SNMP_Research_UNIX_agent.9.4.0.3
UpTime: 4 days, 15 hours, 56 minutes, 39 seconds, 74 hundredths
Contact: Operations 1-999-555-1040
Name: bach.CS.ESU.EDU
Location: Comp Center
Services: applications, end-to-end
```

Removing an incorrect routing entry

Suppose that due to a configuration error, a system in another part of the building had an incorrect entry in its routing table, and that this was causing problems communicating with other hosts in other company offices. The network administrator could use SNMP commands to correct the incorrect routing entry remotely. Example 11-11 shows a possible scenario. In the text, the command line is split with a backslash for clarity. It should be typed in as one line to the shell.

In addition to the new next hop, the administrator specifies the type of route. In this case, the route is an indirect or remote route (type 4), which was reported by **snmpstat** as **rem**.

Example 11-11 Correcting routing entries

```
# snmpstat -r grinch public | grep 123.0.0.0
123.0.0.0         fonebone          2       rem     rip       ni0

# setany grinch public ipRouteNextHop.123.0.0.0 \
-a 128.212.64.1 ipRouteType.123.0.0.0 -i 4

Name: ipRouteNextHop.123.0.0.0
Value: 128.212.64.1

Name: ipRouteType.123.0.0.0
Value: 4
```

The administrator could also have deleted the route by setting its **ipRouteType** to invalid (type 2). This is useful if a routing protocol reacquires the route information automatically.

Marking an interface down

Suppose that a PPP interface attached to a router is reporting many errors. The administrator might wish to mark the link down while the phone company checked the line. SNMP could be used to do this remotely.

First, the error count can be obtained with **snmpstat**.

```
# snmpstat -i grinch public
Interface statistics
                        Type        InOctet   InPckts   InErrs   IfMtu
Name    Address         Speed       OutOctet  OutPckts  OutErrs  OutQlen
ni0     grinch          enetv2      18952186  18239     7036     1500
        02000000ec6c    10000000    4392148   4183      147      0
lo0     localhost       loop        5832      64        0        2048
                        0           5832      64        0        0
ppp0    grinch          ppp         3682304   14384     4800     296
                        9600        2122240   8290      2138     0
```

Next, the interface can be marked down using **setany**.

```
# setany grinch public ifAdminStatus.3 -i 2
Name: ifAdminStatus.3
Value: 2
```

When the line is repaired, the interface can be reactivated in a similar fashion.

Removing an incorrect ARP entry

Suppose that a misconfigured system had an incorrect IP address, causing bad ARP information to be installed on another node. SNMP could remove the offending address remotely. This can be very useful if the node in question has a very long timeout on the ARP cache, or some important connection is being established with the wrong machine. Example 11-12 shows a possible scenario.

Example 11-12 Removing ARP entry

```
# snmpstat -a grinch public
Address Translation table
Host                Physical Address      Flags Interface
128.212.64.9        8:0:14:40:1:21        dyna  ni0
128.212.64.79       0:0:c0:6d:82:23       dyna  ni0

# setany grinch public ipNetToMediaType.1.128.212.64.9 -i 2

# snmpstat -a grinch public
Address Translation table
Host                Physical Address      Flags Interface
128.212.64.79       0:0:c0:6d:82:23       dyna  ni0
```

More SNMP information

The following RFC's (Requests for Comments) contain more information about SNMP and other related protocols. To obtain these RFC's, contact Jon Postel, RFC Editor, USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA, 90292-6695.

RFC 768: User Datagram Protocol; J. Postel; 8/80

RFC 1058: Routing Information Protocol; Hedrick, C.L.; 6/88

RFC 1155: Structure and Identification of Management Information for TCP/IP-Based Internets.; Rose, M.T., McCloghrie, K.; 5/90

RFC 1156: Management Information Base for Network Management of TCP/IP-Based Internets; McCloghrie, K., Rose, M.T.; 5/90

RFC 1157: Simple Network Management Protocol (SNMP); Case, J.D., Fedor, M., Schoffstall, M.L., Davin, C.; 5/90

RFC 1213: Management Information Base for Network Management of TCP/IP-based Internets:MIB-II; McCloghrie, K.; Rose, M.T.,eds.; 3/91

Chapter 12

Remote line printing

Remote Line Printing (RLP) allows UNIX system machines on a network to send print jobs to other UNIX system machines that have attached printers. The machines that send print jobs are called clients. The machines that directly control the printers are called print servers. Clients and print servers can be either an SCO system or a 4.3BSD system (or one of its derivatives, such as SunOS™).

The only restrictions on RLP are:

- printer classes are not supported
- not all line printer command options are supported on the client

RLP allows print jobs to be submitted from a client and printed by a print server, but it does not support remote administration of printers. To administer a printer attached to a print server, do one of the following:

- Log in to the print server at the console.
- Log in to the print server from another machine using terminal emulation.
- Issue printer administration commands from another machine using the **rcmd**(TC) command.

Installing and removing RLP

Before installing RLP on your SCO system, be sure first to install TCP/IP. To install RLP, log in as *root*, then enter the command **mkdev rlp**. You now see the following display:

```
        Remote printing configuration

Do you want to install or delete remote printing (i/d/q)?
```

You have three options:

- To quit and return to the shell prompt, enter **q**.
- To remove RLP after it has been installed, enter **d**. Once RLP has been removed, your local printing environment is restored and you are returned to the shell prompt.
- To install or reinstall RLP, enter **i**. Installation details are discussed in the rest of this section.

If you choose to install RLP, you now see the following messages:

```
Installing Remote Line Printing

Creating directories /usr/lpd/remote and /usr/spool/lpd and file /etc/printcap
Saving lp, cancel and lpstat commands to /usr/lpd/remote
Installing remote lp, cancel and lpstat commands in /usr/bin
Do you want to change the remote printer description file /etc/printcap (y/n)?
```

To define printers available through RLP, enter **y**. The **rlpconf** command is then executed. See "Setting up a client" and "Setting up a print server" later in this chapter for information on **rlpconf**.

To wait until later to define printers, enter **n**. You then see the message:

```
Run /etc/rlpconf to change the remote printer description file
```

Remember to run **rlpconf** at some point before using RLP. Otherwise, you will have no entries in the */etc/printcap* file.

If the directories that RLP needs to do the installation already exist, you are asked the following:

```
Remote line printing working directory already exists.

Do you wish to continue installing the remote line printing system (y/n)?
```

Enter **y** to continue the installation. Enter **n** to stop and return to the shell prompt.

Next you see the following message:

```
Setting up rc scripts
```

This message tells you that the appropriate scripts are being defined to start up RLP every time the system is booted.

You are now asked the following question:

```
Do you want to start remote daemon now (y/n)?
```

To start the **lpd**(ADMN) daemon, enter **y**. This daemon must be running for RLP to work. To wait until later to start the daemon, enter **n**. See the manual page on **lpd** for more information.

How RLP works

RLP operates somewhat differently from normal printing. First, a new file, called */etc/printcap*(SFF), contains information about local and remote printers and related files needed by the print commands. Second, the usual SCO print commands (**lp**, **cancel**, and **lpstat**) are moved to the directory */usr/lpd/remote* for safekeeping, and new versions of these command are moved to */usr/bin*. These new commands consult the file */etc/printcap* to determine whether the print job is going to a local printer or a remote printer attached to a print server.

If the printer is local, printing occurs just as it would with the standard command. All standard command options are supported. This is because the new command simply invokes the standard command, now located in */etc/lpd/remote*. If the printer is remote, the command spools the print job and invokes the **lpd** daemon. This daemon packages the print job in 4.3BSD format and sends it over the network to the host specified in the */etc/printcap* file.

Using RLP

To use RLP, you invoke one of the supported commands. There are actually two sets of commands: one for SCO clients and one for 4.3BSD clients and derivatives.

SCO clients

On SCO clients, the RLP commands are **lp, cancel,** and **lpstat.** These commands work just like their non-RLP counterparts when invoked with no options and when invoked with the following supported options:

lp -d*printer_name*
lp -n*number_of_copies*
lp -t*title_to_print_on_banner*

lpstat -a*remote_printer_name*	(acceptance status)
lpstat -o*remote_printer_name*	(output requests status)
lpstat -p*remote_printer_name*	(printer status)
lpstat -v*remote_printer_name*	(pathnames of associated devices)

Only the options listed above are supported for the **lp** and **lpstat** commands when used for remote printing. If one or more unsupported options are specified, they are ignored and the print job is submitted without them. If printing is to be done locally, all of the options that are normally available for local printing are supported.

NOTE On an SCO client, you must install RLP before you can do remote printing. See "Installing and removing RLP" earlier in this chapter for more information.

See the **cancel**(C), **lp**(C), and **lpstat**(C) manual pages for more information on the supported options.

4.3BSD clients

On 4.3BSD clients and derivatives, use the standard printing commands **lpr**, **lprm**, and **lpq** to do remote printing. The following options are supported when sending a print job to a remote printer attached to an SCO system on which RLP has been installed:

lpr -P*printer_name* [-#*number_of_copies*] [-T*title*] [-s]

(The **-s** option uses symbolic links instead of copying the file to the spool directory.)

lpq -P*printer_name*

lprm -P*printer_name*
lprm -P*printer_name job_number*

Only the options listed previously are supported for the **lpr**, **lpq**, and **lprm** commands when used for remote printing. If one or more unsupported options are specified, they are ignored and the print job is submitted without them.

NOTE If the remote printer is attached to a print server running 4.3BSD or a derivative, the above restrictions do not apply. In this case, both the client and the print server are running 4.3BSD or a derivative. Remember that RLP is installed only on SCO systems.

If printing is to be done locally, all of the options that are normally available for local printing are supported.

Setting up a client

The printer description file */etc/printcap* on the client machine provides routing and other information about printers.

NOTE If the printer to be set up is a local printer, that is, attached to the client for its private use, you should set it up following the instructions in "Setting up a print server" later in this chapter, but ignore instructions regarding the */etc/hosts.equiv* and */etc/hosts.lpd* files.

For a full description of the */etc/printcap* file format, see the *printcap*(SFF) manual page.

Here is an example of an entry in the */etc/printcap* file, followed by an explanation of each field. Note that fields are separated by colons. Also, this entry is actually one logical line continued over two lines on the screen with the continuation character " \ ".

```
sunlaser:\
:lp=:rm=sunburst:rp=sunlaser:sd=/usr/spool/lpd/sunlaser
```

Each of the fields is described below:

1. The first field in this example is *sunlaser*, the name to be used on the client when referring to this printer. If the printer is remote, the name must be the same one used by the print server.
2. The second field (before the first colon on the second line) is empty by default. When used, it defines the name of the error log file.
3. The third field, starting with **lp=**, must be empty (that is, followed immediately by a colon) for remote printing. This field specifies the device name for a local printer.
4. The fourth field, starting with **rm=**, specifies the network name of the print server. In this example, the print server is called *sunburst*.
5. The fifth field, starting with **rp=**, specifies the name that the print server uses for the printer.
6. The last field, starting with **sd=**, is the name of the spooling directory on the client. It is in this directory that print jobs are stored before being sent over the network to the print server. This directory is always located in */usr/spool/lpd* and bears the name of the printer. In this example, the spool directory is called */usr/spool/lpd/sunlaser*.

To configure a remote printer, execute the **rlpconf** command. This command edits the */etc/printcap* file for you. The following example illustrates the use of **rlpconf** to set up the *sunlaser* printer to produce an entry in the */etc/printcap* file like the one shown at the beginning of this section. Assuming you are logged in as *root*, invoke the **rlpconf** command. You see output similar to the following:

```
                Remote Printing Configuration

Enter information for remote printers or local printers accepting
remote print jobs.

Please enter the printer name (q to quit): sunlaser

Is sunlaser a remote printer or a local printer (r/l)? r

Please enter the name of the remote host that sunlaser is
attached to: sunburst

        Printer sunlaser is attached to host sunburst

Is this correct? (y/n) y
Would you like this to be the system default printer? (y/n) y

Make sure that your host name appears in sunburst's /etc/hosts equiv
or /etc/hosts.lpd file.
Make sure that sunlaser appears in sunburst's /etc/printcap file (BSD format)
Make sure that sunlaser has a spool directory on sunburst
Putting sunlaser in printer description file and creating spool dir
Updating LP information...done
Updating /usr/spool/lp/default...done
```

The above example shows how to configure your client to use a remote printer called *sunlaser* attached to the print server *sunburst*. After entering the name of the printer and the print server, you must verify the information just entered. If it is correct, enter **y**. If it is not, enter **n**, at which point you are asked if you want to try again. Answer **y** to be prompted again for the name of the printer and the print server. Answer **n** to exit and return to the shell prompt.

Continuing with the above example, you are asked if you wish to make the printer the system default. Answer **y** or **n**. If you answer **n**, users on the client that wish to route a print job to *sunlaser* must do so explicitly by using the **-d** option in the **lp** command.

Following this query, you are advised to verify certain things on the print server. The **rlpconf** command now checks that the printer is not already listed in */etc/printcap*. If it is, a message to that effect is displayed. If it is not, the printer information is added to */etc/printcap*.

In addition, the spool directory for the printer, in this instance */usr/spool/lpd/sunlaser*, is created and configuration information indicating the name of the printer and the print server is stored in a file called *configuration*. This file is located in a subdirectory bearing the printer's name under */usr/spool/lp/admins/lp/printers*. In this example, the pathname of the file is */usr/spool/lp/admins/lp/printers/sunlaser/configuration*. Finally, if the printer is to be the system default, the file */usr/spool/lp/default* is updated to contain the name of the printer.

Once you have configured the printer, the screen clears and you see output similar to the following:

```
                Remote Printing Configuration

Enter information for remote printers or local printers accepting
print jobs.

Please enter the printer name (q to quit): q
#
```

Enter **q** to exit **rlpconf** and return to the shell prompt.

Once you have finished configuring printers, ensure that the RLP daemon **lpd** is running. You can do this by entering the command **ps -fe** and verifying that the daemon appears in the listing.

Setting up a print server

To set up a print server to process RLP requests, do the following:

- Ensure that the RLP daemon **lpd** is running. You can do this by entering the command **ps -fe** and verifying that the daemon appears in the listing.
- Ensure that that all printers attached to the print server are working properly as local printers. See the documentation that accompanies the operating system running on the print server for information on how to set up local printer services.
- List all of the clients that are to have access to the print server in either the */etc/hosts.equiv* or the */etc/hosts.lpd* file using your favorite text editor. Listing a client in the latter file is more secure, in that only remote printer access is granted to the specified client. See the manual page that describes */etc/hosts.equiv* for information on the kind of access made available to hosts listed in that file.
- Execute **rlpconf** to create the */etc/printcap* file and the spool directories for printers attached to the print server.

Once you invoke **rlpconf**, you see output similar to the following:

```
                Remote Printing Configuration

Enter information for remote printers or local printers accepting
print jobs.

Please enter the printer name (q to quit): sunlaser

Is sunburst a remote printer or a local printer (r/l)? l

Please enter the name of the device for sunlaser: /dev/laser

        Printer sunlaser uses device /dev/laser

Is this correct? (y/n) y

Putting sunlaser in printer description file and creating spool
directory...done
```

First enter the name of a printer attached to the print server. Next, enter **l** in response to the second question to indicate that you are on a print server. Then, enter the full pathname for the device for the printer. You are now given the opportunity to verify the device filename for the printer. If it is correct, enter **y**. If it is not, enter **n**, at which point you are asked if you want to try again. Answer **y** to start from the beginning or answer **n** to exit and return to the shell prompt.

Once you indicate that the device filename is correct, the **rlpconf** command checks that the printer is not already listed in */etc/printcap*. If it is, a message to that effect is displayed. Otherwise, the printer information is added to the */etc/printcap* file and the spool directory for the printer, */usr/spool/lpd/sunlaser*, is created.

Once processing is complete for one printer, you are prompted to set up another printer. When you are done, enter **q** to exit **rlpconf** and return to the shell prompt.

Deleting printcap entries

To delete an entry in the */etc/printcap* file, you must use your favorite text editor. Be sure to delete the entire entry, which is continued over two lines. Also, you should delete the spool directory */usr/spool/lpd/**printer_name*** and the file */usr/spool/lp/admins/lp/printers/**printer_name**/configuration* associated with the deleted printer. Finally, if the deleted printer is the system default printer, you should modify the file */usr/spool/lp/default* appropriately. See "Setting up a client" earlier in this chapter for more information.

Chapter 13

Synchronizing clocks

Making sure that all of the clocks running on the various hosts in a network or group of interconnected networks keep accurate time is a very important task. One reason is that many programs, including database management systems and configuration management systems, use timestamps as part of their processing. This task, often referred to as time synchronization, is much more difficult to achieve in the distributed processing environments often found in networks. Both the Time Synchronization Protocol (TSP) and the Network Time Protocol (NTP) can be used to achieve time synchronization.

Only one of these protocols should be used on a given network or group of interconnected networks. Using NTP is preferred because it keeps clocks synchronized with Coordinated Universal Time (UTC), the international time standard. To use NTP on your network, you must have access to the Internet. If you do not, your only alternative is to use TSP. Unlike NTP, TSP does nothing more than attempt to achieve local synchronization of clocks. In other words, it seeks to have all clocks on the locally interconnected networks keep the same time, but it does not attempt to keep clocks synchronized to an external, standard time reference. Each protocol is discussed in this chapter in a separate section.

Time synchronization protocol

In this implementation of the Time Synchronization Protocol, the **timed**(ADMN) time daemon runs on each host on your network. These time daemons communicate with each other using a master-slave scheme in which one of the daemons is the master and all of the remaining daemons are slaves.

Each time daemon has two functions:

- it supports the synchronization of the clocks on the various hosts in the network
- it starts or takes part in the election of a new master if the current master becomes unavailable or stops functioning

The synchronization mechanism and the election procedure are described on the **timed** manual page. The rest of this section describes the administrative and technical issues of running **timed** at a particular site.

How the time daemon works

A master time daemon measures the time differences between the clock of the host on which it is running and those of all the other hosts on the network. The master computes the network time as the average of the times provided by slaves on hosts whose clocks are not faulty. In this scheme, a host's clock is considered faulty if its time differs from that of the majority of the clocks on the network by more than a specified amount. The master time daemon then sends to each slave time daemon the correction that should be applied to its host's clock. This process is repeated periodically.

Because the correction is expressed as a time difference rather than an absolute time, transmission delays do not interfere with the accuracy of the synchronization. Whenever a host comes up on the network, it starts a slave time daemon that asks the master for the correct time and resets the host's clock before any user activity can begin. The time daemons are thus able to maintain a single network time in spite of the drift of clocks away from each other. The present implementation of **timed** is capable of keeping processor clocks synchronized to within 20 milliseconds, but some clocks, because of their design, cannot be adjusted to within an interval smaller than 1 second.

To ensure that the time synchronization service provided is continuous and reliable in the face of perturbing events, an election algorithm is available to elect a new master. Perturbing events include:

- the host running the current master time daemon crashes
- the master time daemon terminates because of a runtime error
- the network on which the master time daemon is running is partitioned or restructured

This algorithm allows slaves to detect that a master has stopped functioning and to elect a new master from among the slaves. Because the failure of a master time daemon results in only a gradual divergence of clock times among the slaves, the election of a new master need not occur immediately.

Running **timed** on gateways that connect distinct Local Area Networks (LANs) requires particular care because the time daemon may act as a "submaster". A submaster is a time daemon that functions as a slave on one of the networks the gateway is connected to, and as a master on one or more of the remaining networks. Submasters are necessary to overcome the restriction of current transmission protocols preventing broadcasts from being transmitted over multiple physical networks.

A submaster classifies the networks that the gateway is connected to into the following three types:

- a slave network, which is a network on which the submaster acts as a slave. Only one of the networks can be a slave network.
- master networks, which are networks on which the submaster acts as a master
- ignored networks, which are networks that are neither slave networks nor master networks

The submaster tries periodically to become master on ignored networks, but gives up immediately once it detects that a master already exists on that network.

Guidelines

While the synchronization algorithm is quite general, the election algorithm, which requires a broadcast mechanism, puts constraints on the kind of network on which **timed** can run. One restriction is that you should run **timed** only on networks with broadcast capability, possibly augmented with point-to-point links. Another restriction is that the maximum number of hosts that can be directly controlled by one master time daemon is 99.

If submasters are excluded, there is normally only one master time daemon running on a given LAN. During an election, only one of the slave time daemons becomes the new master. Not all hosts are suitable as masters, however. Because a master time daemon requires CPU resources proportional to the number of slaves it controls, you should consider running a master only on hosts with more powerful processors or lighter processing loads. Also, hosts with inaccurate clocks should not be used as masters. Note that this is a purely administrative decision. You are free to allow all of the hosts on your network to be a master.

At startup time, a time daemon running on a host connected to multiple networks may be instructed to recognize specific networks with the **-n** option or to ignore specific networks with the **-i** option. Note that both options cannot

be specified together. In other words, you can recognize certain networks or ignore certain networks, but you cannot do both. Typically, you would instruct the time daemon to ignore all networks except those that are under local administrative control.

If you run **timed** on several different LANs connected over gateways, you should avoid the situation where two gateways are both connected to the same two networks, and where the time daemon on those gateways play opposite master/slave roles on those networks. For example, suppose hosts A and B are gateways that are both connected to networks X and Y. If **timed** is started on both A and B with the **-M** option, it is possible for the time daemon on gateway A to become a slave on network X and the master on network Y, while the time daemon on gateway B can become a slave on network Y and the master on network X. If this kind of loop exists, the master time daemons do not function properly to establish a single value for time on both of the networks. It also causes the submasters to use large amounts of system resources in the form of network bandwidth and CPU time. This kind of loop can also be generated with more than two gateways, in the case where several LANs are interconnected.

Options

The options for the **timed** command are:

-n ***network***	recognize the named network
-i ***network***	ignore the named network
-t	place tracing information in */usr/adm/timed.log*
-M	allow this time daemon to become a master. A time daemon started without this option will always be a slave.

Daily operation

As part of the day-to-day operation of your network, you can use the **timedc**(ADMN) command to control the operation of the time daemons. For example, you can

- measure the differences between host clocks
- find the location of the master time daemon
- cause election timers on several hosts to expire at the same time
- enable or disable tracing of messages received by **timed**

Also, you can set the network date using the **rdate**(ADMN) command. See the manual pages on **timed**, **timedc**, and **rdate** for more information.

Network time protocol

This section describes the Network Time Protocol (NTP) and offers guidelines on how to configure NTP. It includes example configurations, and describes how to get information on how well NTP is working, how to test NTP, how to tune it, and how to troubleshoot error conditions.

Important terms

A number of terms describe important entities or activities that are part of NTP. These are defined here.

client mode — the mode in which a host polls a time server that it might synchronize with, but it will not respond to polls from that time server. When a host is operating in this mode, the time server it is polling is said to operate in "server mode".

Coordinated Universal Time — the international standard reference time. It also corresponds to the local time at zero longitude. The standards pertaining to the definition and maintenance of Coordinated Universal Time, which effectively replaced Greenwich Mean Time (GMT) on 1 January 1972, are promulgated in Recommendation 460 of the International Consultative Committee for Radio (Comite Consultatif International de Radiodiffusion or CCIR) of the International Telecommunications Union. The CCIR is located at 2, rue de Varembe, 1211 Geneva 20, Switzerland. Responsibility for time standards in the United States rests with the Time and Frequency Division of the National Institute of Standards and Technology (NIST) in Boulder, Colorado.

dispersion — a measure, in seconds, of how scattered the time offsets have been from a given time server

drift — a measure, in Hertz per second, of how quickly the skew of a clock is changing. See also "skew".

host — any computer connected to the network

Internet — the collection of interconnected networks that grew out of ARPANET, the Defense Advanced Research Projects Agency network, and that use TCP/IP to communicate to function as a single cooperative network

poll — the sending of an NTP packet from a host to an NTP time server to request the current time. The server responds by recording the current time in the packet, then sending it back to the originating host. See also "NTP packet."

NTP packet
: a message sent over the network that conforms to the Network Time Protocol format. This format includes space for recording the current time. See also "poll".

primary server
: another name for a stratum 1 server. See also "stratum".

roundtrip delay
: the time it takes for a host to send an NTP packet to another host and get an NTP packet back from that host in reply

secondary server
: another name for a stratum 2 server

server mode
: the mode in which a time server allows itself to be polled by a host (the client) that wishes to synchronize with it. In this mode, if the time server polls the client to try to synchronize with it, the client does not respond. In this case the client is said to operate in "client mode".

skew
: a measure, in Hertz, of the difference between the actual frequency of a clock and what its frequency should be to keep perfect time. See also "drift".

slew
: to adjust gradually the time of a clock until it tells the correct time. Compare with "step".

step
: to change the time of a clock to the correct time with no intermediate adjustments. Compare with "slew".

stratum
: the distance a host running the **xntpd** time daemon is from an external source of Coordinated Universal Time (UTC). A stratum 1 server has direct access to an external source of UTC, such as a radio clock synchronized to a standard time signal broadcast. In general, a stratum ***n*** server is ***n-1*** network hops away from a stratum 1 server. For example, a stratum 4 server is 3 hops away from a stratum 1 server. Also, a stratum ***n*** server is at a higher stratum than a stratum ***n-1*** server. For example, a stratum 3 server is at a higher stratum than a stratum 2 server, and at a lower stratum than a stratum 4 server. See also "time daemon".

symmetric active mode
: the mode in which a host has configured itself to poll a time server that it might synchronize with. In this mode, the host also allows itself to be polled by that time server.

symmetric passive mode
: the mode in which a time server is polled by a host that has configured itself in "symmetric active mode". In this mode, the time server can also poll that host.

synchronization subnet
: a collection of hosts that synchronize time with each other. The top layer of the subnet consists of stratum 1 servers.

synchronize clocks
: to set two clocks to the same time and ensure that they are running at the same "speed". The speed at which a clock runs is determined by its frequency, that is, how often it "ticks" to the next fraction of a second. The design of a particular clock determines how small that fraction is.

synchronize with a host
: to synchronize the local clock with another host's clock, either by stepping or slewing the local clock to the time reported in the NTP packet received from the most recent poll of that host.

time client
: a host running the **xntpd** time daemon that requests a time server to furnish it with that server's best estimate of Coordinated Universal Time. Hosts running at a stratum higher than 1, except for those at the highest stratum, typically function as both time clients (polling same-stratum or higher-stratum servers) and time servers (furnishing the current time to other hosts). Compare with "time server".

time daemon
: the program running on a host that synchronizes the host's hardware clock to Coordinated Universal Time in accordance with the protocols known as the Network Time Protocol. The name of this program is **xntpd**.

time server
: a host running the **xntpd** time daemon that, upon request, furnishes its best estimate of Coordinated Universal Time. Compare with "time client".

trap receiver
: a program that listens for NTP mode 6 packets sent from another host. The host that "springs the trap" by sending the mode 6 packet to the trap receiver does so in response to the occurrence of an exception. These exceptions include the following:

  - host reboots
  - host detects that its clock has been reset
  - host detects that the stratum of some other host in the synchronization subnet has changed
  - host detects that a new host has been added to the synchronization subnet
  - host detects that some other host in the synchronization subnet has become unreachable

- host detects that some other host in the synchronization subnet has been authenticated
- host selects another host to synchronize with
- host detects that a system call has failed

Overview

The Network Time Protocol (NTP) defines a set of procedures for synchronizing clocks on hosts connected to a network with access to the Internet. Some of the hosts act as time servers, that is, they provide what they believe is the correct time to other hosts. Other hosts act as clients, that is, they find out what time it is by querying a time server. Some hosts act as both clients and time servers, because these hosts are links in a chain over which the correct time is forwarded from one host to the next. As part of this chain, a host acts first as a client to get the correct time from another host that is a time server. It then turns around and functions as a time server when other hosts, acting as clients, send requests to it for the correct time.

The network time daemon **xntpd**(ADMN) is the program running on each of the hosts in the network that, following the conventions defined in the protocol, attempts to establish the correct time. It does this by using the best available source of time to synchronize the host clock with the time at zero longitude, known as Coordinated Universal Time or UTC.

Guidelines

If your network is small, consisting of around a dozen hosts, you need only a single NTP time server on your network. Locate three time servers elsewhere on the Internet from which your time server can get the correct time. If possible, pick two stratum 1 servers and one stratum 2 server. Ideally, you should choose to get the time from servers that are closest to your host in terms of roundtrip delay. In other words, you should prefer servers for which the amount of time it takes to send a request for the current time and the time it takes to receive a response is smallest. However, there is no great harm if you select servers for which you do not know the roundtrip delay.

If, however, your network is large, consisting of dozens or hundreds of hosts, you should set up a larger number of time servers. A good rule of thumb is to have one time server for every dozen hosts. In this case, you first establish a few stratum 2 time servers, then you establish the remaining time servers as stratum 3 time servers. Finally, all of the remaining hosts, which constitute the vast majority of the hosts on the network, become stratum 4 hosts.

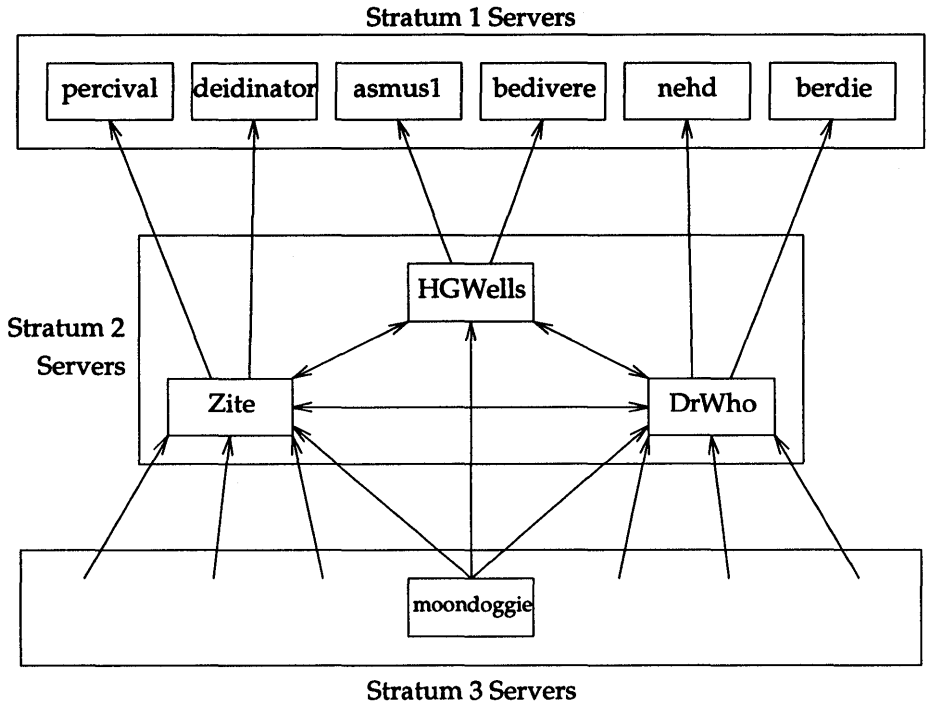
Below are guidelines for setting up NTP on a network of medium size.

- Choose three of your local hosts to operate as stratum 2 servers. Then choose six stratum 1 servers and configure each of your stratum 2 servers to poll a distinct pair of the stratum 1 servers. If possible, see to it that each stratum 2 server polls at least one stratum 1 server for which the roundtrip delay is small. You should also configure each of the stratum 2 servers to poll both of the other stratum 2 servers. Thus, each stratum 2 server is configured to poll two stratum 1 servers elsewhere on the Internet and two stratum 2 servers connected to the local network.
- From the remaining hosts, choose those you would like to operate at stratum 3. This might be all of the rest if you want to synchronize at most a couple of dozen hosts, or you might choose hosts such as file servers or those with good clocks if you need to synchronize many more hosts. Configure each stratum 3 server to poll the three stratum 2 servers.
- Configure all remaining hosts to either poll or listen for broadcast packets from three or four of the nearby stratum 3 servers.
- Avoid configuring hosts beyond stratum 4. Doing so makes the network unnecessarily complex and harder to manage.
- Do not configure a host to poll another host at the same stratum, unless the latter is directly polling lower stratum servers that the former does not.
- Do not configure hosts to poll other hosts at higher strata.

An example synchronization subnet

Assume that hosts *Zite*, *HGWells*, and *DrWho* are stratum 2 servers in the imaginary *sci.com* network. Assume also the following:

- *Zite* polls two stratum 1 servers, *percival.cs.purdue.edu* and *deidinator.psyc.chi.il.us*.
- *HGWells* polls two stratum 1 servers, *asmus1.genetics.uga.edu* and *bedivere.cs.purdue.edu*.
- *DrWho* polls two stratum 1 servers, *nehd.gov* and *berdie.athens.ga.us*.
- *Zite* polls *HGWells* and *DrWho*.
- *HGWells* polls *Zite* and *DrWho*.
- *DrWho* polls *Zite* and *HGWells*.
- *Moondoggie* and the rest of the machines at *sci.com* are at stratum 3 and listen for broadcast NTP packets from the three stratum 2 hosts.



A → B means A polls B or A listens for broadcast NTP packets from B.
C ⟷ D means C polls D and vice versa.

The NTP configuration file

The configuration file */etc/ntp.conf* contains information that the **xntpd** daemon uses at startup, including:

- what are the possible hosts to synchronize with
- how to select which host to synchronize with
- what restrictions exist on other hosts communicating with the local host
- whether or not to broadcast NTP packets on the local network
- whether or not to listen for broadcast NTP packets
- what the roundtrip delay is between the local host and the host on this network (if any) that is broadcasting NTP packets
- where the *driftfile* is located
- whether or not to monitor NTP connections
- whether or not to allow runtime reconfiguration of the local host

The syntax of the configuration file is fairly free format. Comments begin with a "#" character and extend to the end of the line. Comments may be inserted freely and blank lines are ignored. Configuration statements include an initial keyword followed by arguments that are separated by white space (blanks or tabs). These statements may not be split over multiple lines. A list of the configuration statements with their arguments appears below. Optional arguments are delimited by "[" and "]", and alternatives are separated by " | ".

Configuration statements

Below is a list of the configuration statements you can use in the NTP configuration file and an explanation of each statement. The term "local host" in the explanations refers to the host that you are configuring with these statements.

peer ***host_address*** **[key** ***key_ID*** **] [version** ***version_number*****] [minpoll]**

specifies that the host with IP address ***host_address*** is to be polled in symmetric active mode. The **key** option indicates that all packets sent to the host at ***host_address*** are to include authentication fields encrypted using the specified key ID. The default is not to include authentication fields. The **version** option allows you to specify the version number to use for outgoing NTP packets. The allowed values for ***version_number*** are **1** and **2**. If this option is omitted, version 2 is assumed. The **minpoll** option specifies that the polling interval should be kept at the minimum value allowed. This maximizes the frequency with which the host is polled and should, therefore, be used only for debugging purposes.

server ***host_address*** **[key** ***key_ID*** **] [version** ***version_number*****] [minpoll]**

specifies that the host with IP address ***host_address*** is to be polled in client mode. See the **peer** statement for an explanation of the remaining arguments.

broadcast ***IP_address*** **[key** ***key_ID*** **] [minpoll]**

specifies that **xntpd** broadcast NTP packets to the address ***IP_address***. This should be the broadcast address of your local network. See the **peer** statement for an explanation of the remaining arguments.

precision ***integer***

indicates the precision of the local clock as 2 raised to the ***integer*** power. The value of ***integer*** should be negative to indicate a fraction. See "Testing and tuning" later in this chapter for more information.

driftfile ***filename***

specifies the name of the file used to record the drift that **xntpd** has computed. Drift is also known as "frequency error".

monitor ***flag***

indicates whether or not to enable the **xntpd** monitoring function. The allowed values for ***flag*** are **yes** and **no**. A valid **requestkey** statement must be included in the configuration file for monitoring to function. The default value for ***flag*** is **no**.

broadcastclient ***flag***

indicates whether or not the local host should listen for, and attempt to synchronize with, NTP packets broadcast by time servers on the local network. The allowed values for ***flag*** are **yes** and **no**. The default is **no**.

broadcastdelay ***seconds***

specifies the roundtrip delay in seconds between the local host and the time server that is broadcasting NTP packets. Note that you specify a roundtrip delay, even though NTP cares only about one-way delay when it actually computes the correct time. The default value for ***seconds*** is 0.008.

keys ***pathname***

specifies the pathname of the *keys* file. At installation the value of ***pathname*** is */etc/ntp.keys*. This file must exist for the **requestkey** and **controlkey** configuration statements to be valid. See "The keys file" later in this chapter for more information.

requestkey ***key_ID*** ...

specifies the key IDs used to authenticate runtime reconfiguration requests made by **xntpdc** and **xntpres** using mode 7 control messages. The *keys* file must exist and properly define all of the listed key IDs for this statement to be valid. To disable runtime configuration of **xntpd**, omit this statement from the configuration file. Doing so has the effect of disabling **xntpdc** and **xntpres** on your local host.

controlkey ***key_ID*** ...

specifies key ID used to authenticate runtime reconfiguration requests made by **ntpq** and by trap receivers using mode 6 control messages. The *keys* file must exist and properly define all of the listed key IDs for this statement to be valid. To cause mode 6 control messages to be ignored, omit this statement from the configuration file. Doing so has the effect of disabling **ntpq** and trap receivers on your local host.

restrict ***host_address*** **mask** ***address_mask*** **flag** ...

specifies various restrictions on how certain hosts and networks can interact with the local host. See "Address and mask facility" later in this chapter for more information.

trap ***host_address*** **port** ***port_number*** **interface** ***interface_address***

configures a trap receiver at address ***host address*** and port ***port number***. The value of ***interface_address*** must be the IP address of the local host. The host at ***host_address*** must be listening for NTP mode 6 control messages or it will not respond to the trap signal sent by the local host. This means that the host at ***host_address*** must have the **controlkey** statement in its configuration file and both that host and the local host must share at least one of the keys listed in that **controlkey** configuration statement. See "The keys file" later in this chapter for more information.

maxskew ***seconds***

sets the system maximum skew parameter to the number of seconds specified. See "Testing and tuning" later in this chapter for more information.

select ***algorithm_number***

specifies one of five weighted algorithms (numbered 1 through 5) used to select which time server the local host synchronizes with, out of the set of time servers from which the local host is receiving NTP packets. The default value for ***algorithm_number*** is **1**.

All of the algorithms attempt to select an accurate source time source (a "truechimer") without excessive hopping from one time source to another. These algorithms take into account the following pieces of information:

- stratum of time source
- transmission delay between host and time source
- offset of clock on time source
- dispersion of clock on time source

In comparison to algorithm 1, algorithm 2 gives more weight to low-stratum time sources (algorithm 3 even more so), whereas algorithm 4 gives more weight to time sources for which the roundtrip delay is small (algorithm 5 even more so).

The basic algorithm is described in RFC 1119 Section 4 and updated in RFC 1129 Section 4.2. The weights associated with the different algorithms are: 0.75 for algorithm 1; 0.6875 for algorithm 4; and 0.625 for algorithm 5. Compared to algorithm 1, algorithm 2 reduces the weight of high stratum hosts by about 9%, while algorithm 3 reduces the weight of high stratum hosts by about 18%.

resolver ***pathname***

indicates the full pathname of the **xntpres** program, which resolves network names to IP addresses. When NTP is installed,

the pathname of the **xntpres** program is */etc/xntpres*. If for some reason you move **xntpres** to another directory, use the pathname corresponding to its new location. You must also include the **requestkey** statement in the configuration file for **xntpres** to operate.

Example ntp.conf file

The following is an example of a minimal *ntp.conf* file. The name of the host in this example is *zite.sci.com*.

```
#
# Peer configuration for 128.212.66.90 (zite.sci.com)
# (expected to operate at stratum 2)
#
peer 128.211.1.4      # percival.cs.purdue.edu (stratum 1 server)
peer 192.35.53.67     # deidinator.psyc.chi.il.us (stratum 1 server)

peer 128.212.66.67    # HGWells.Sci.com
peer 128.212.66.118   # DrWho.Sci.com

driftfile /etc/ntp.drift
```

The four **peer** statements tell **xntpd** which servers to poll. The **driftfile** statement tells **xntpd** the name of the file in which to store the frequency error of the system clock. You are strongly encouraged to include this statement in the configuration file. This server is expected to operate as a stratum 2 server, because the first two servers have hardware clocks and are, therefore, stratum 1 servers.

The keys file

The *keys* file contains a list of numeric key IDs and key values. These IDs and values are used to verify that mode 6 and mode 7 NTP packets should be processed. For example, when running the **xntpdc** program, you must supply a valid key ID in response to the `Keyid` prompt and its associated key value in response to the `Password` prompt. See "Further examples" later in this chapter for sample displays of this.

In addition to a key ID and its associated value, each entry also contains a one-letter code indicating the type of the key value. The precise format of an entry in the key file is:

```
key_ID      key_type      key_value
```

The three fields shown above are separated by any combination of blanks and tabs. Comments may appear on any line and must begin with the number sign "#". The fields are:

- key ID, which is an arbitrary, unsigned 32-bit number, written in decimal. The range of possible values is zero through 4,294,967,295. Key IDs are specified by the **requestkey** and **controlkey** statements in the configuration file.
- key type, which identifies the type of ***key_value***. There are three key types:
  - The first type is an ASCII character string between 1 and 8 characters long. The 7-bit ASCII representation of each character is used for each octet of the key. In this format, the high-order bit of each octet is the parity bit. The one-letter code for this type is "A".
  - The second type is a hexadecimal number written in the Department of Defense National Security Agency (NSA) Data Encryption Standard (DES) standard format. In this format, the low-order bit of each octet is the parity bit. The one-letter code for this type is "S".
  - The third type is a hex number written in NTP standard format. In this format, the high-order bit of each octet is the parity bit. The one-letter code for this type is "N".
- key value, which is a 56-bit DES encryption key that is written as 8 octets. The key value always uses odd parity.

Note that you will find it easier to specify ASCII key values, particularly for keys that are used to verify **xntpdc** requests. Because this file contains authorization data, you are strongly urged to limit **read** permission for this file. In particular, you should remove **read** permission for **other**.

Below is a sample *keys* file:

```
#
# Keys 4, 6 and 22 are actually identical at the bit level.
#
4       A    DonTTelL
6       S    89dfdca8a8cbd998
22      N    c4ef6e5454e5ec4c

#
# The following three keys are also identical at the bit level.
#
100     A    SeCReT
1000    N    d3e54352e5548080
100000  S    a7cb86a4cba80101
```

The clock.txt file

The file */pub/ntp/doc/clock.txt* on host *louie.udel.edu* is your basic source of information on the location of stratum 1 and good quality stratum 2 servers. Among other things, this file also lists:

- sources for timecode receivers
- sources for precision oscillators and timing receivers
- call letters, location, frequencies, and coordinates for timecode transmitters
- an algorithm for great-circle distance and bearing computation
- information on timecode transmitters in Germany, the United Kingdom, and France

The information on primary and secondary servers in *clock.txt* has the following format:

Hostname **(*IP address*)**
`Location:` ***organization, geographic location***
`Synchronization:` ***machine type, clock type, synchronization protocol***
`Service area:` ***networks served***
`Access policy:` ***open or closed to the public***
`Contact:` ***name and email address of contact person***
`Note:` ***miscellaneous information***

One way to get the *clock.txt* file is to use **ftp**(TC). Below is a sample session showing how you might download this file to your host.

```
ftp louie.udel.edu
Connected to louie.udel.edu
220 louie.udel.edu.
Name (louie.udel.edu:muckel): anonymous
331 Guest login ok, send ident as password.
Password: guest
230 Guest login ok, access restrictions apply.
ftp> cd /pub/ntp/doc
250 CWD command successful.
ftp> get clock.txt
200 PORT command successful.
150 ASCII data connection for clock.txt (128.212.66.118,1198) (57002 bytes).
local: clock.txt remote: clock.txt
58409 bytes received in 8 seconds (7.1 Kbytes/s)
ftp> quit
221 Goodbye.
```

The driftfile

The **driftfile** entry in */etc/ntp.conf* tells **xntpd** the name of the file where it can find and store the clock drift, also known as frequency error, of the system clock. If the file exists at startup, it is read and the value used to initialize **xntpd**'s internal value of the frequency error. The file is updated once every hour by **xntpd**. It usually takes a day or so after the daemon is started to compute a good estimate of the clock drift. Once the initial value is computed it will change only by relatively small amounts during the course of continued operation. Because **xntpd** needs a good estimate to synchronize closely to its server, there should always be a **driftfile** entry in */etc/ntp.conf*.

Association modes

There are several different modes that hosts can be directly configured to operate in with respect to synchronization with other hosts: client mode, broadcastclient mode, and symmetric active mode.

In client mode, a host polls other hosts to get the current time. From among all of the hosts polled, the local host selects one with which to synchronize. To configure your host this way, include a **server** statement in your host's configuration file. The name or IP address of each time server to be polled must be specified in the **server** statement.

In broadcastclient mode, a host does no polling at all. Rather, it listens for NTP packets broadcast over the network. From among all of the broadcasting hosts, the local host selects one with which to synchronize. To configure your host this way, include a **broadcastclient yes** statement in your host's configuration file. For this mode to function, the local time servers must be configured in broadcast mode with the **broadcast** configuration statement.

Finally, in symmetric active mode, a host polls other hosts and responds to polls from those hosts. In addition, hosts retain time-related information about the hosts with which they are communicating. To configure your host this way, include a **peer** statement in your host's configuration file. The name or IP address of each time server must be specified in the **peer** statement.

A general guideline is to configure all hosts, except those at the highest strata of the synchronization subnet (those furthest away from stratum 1 servers), to operate in symmetric active mode. For those hosts that are at the highest strata, you have a choice: configure them to operate either in client mode or in broadcastclient mode. You should consider opting for broadcastclient mode if your hosts are on a high-speed LAN that supports broadcasts efficiently, especially if the hosts number more than twenty or so.

If you do opt for broadcastclient mode on those hosts, you must configure the time servers on the local network to be both in broadcast mode (to send broadcast NTP packets on the local network) and in symmetric active mode (to poll hosts at lower strata).

Address and mask facility

The address and mask configuration facility adds various restrictions or erect barriers between your host and other time servers. A typical statement in the configuration file looks as follows:

restrict ***IP_address*** **mask** ***IP_address_mask flag1 flag2*** ...

Each statement adds an entry to an internal list maintained by **xntpd**. Each entry in this list contains the list entry address (the IP address following **restrict**), the address mask, and the flags. Below is a list of all of the flags and their meaning:

ignore	indicates that all packets from hosts matching this entry will be ignored
noquery	indicates that your host will not respond to mode 6 and 7 packets sent from hosts with a matching address
nomodify	indicates that your host will not allow itself to be reconfigured by hosts with a matching address
notrap	indicates that your host will not allow hosts with a matching address to register as a trap receiver
lowpriotrap	indicates that your host will give hosts with a matching address a low priority for the use of traps
noserve	indicates that your host will not give the time to hosts with a matching address
nopeer	indicates that your host will not attempt to get time from any host with a matching address
notrust	indicates that hosts matching this entry, while treated normally in other respects, should not be trusted for synchronization

When **xntpd** receives a packet, it compares the address of the host that sent the packet (the source address) with each entry in the internal list. Whenever the following relation (expressed in C language syntax) is true, a match occurs.

```
(source_address & address_mask) == (list_entry_address & address_mask)
```

In words, the source address and the address mask are logically ANDed together bitwise, the list entry address and the address mask are logically ANDed together bitwise, and the two results compared for equality. If the results are equal, a match has occurred. To establish default restrictions that apply to all hosts for which no match is found, include a statement like the following in the configuration file:

```
restrict default flag1 flag2 ...
```

If a particular source address matches more than one list entry, the entry with the most one bits in the address mask is taken to be the matched entry. If a match is found, flags associated with this entry are returned.

Suppose that you are running **xntpd** on a host with IP address 128.212.66.67. You would like to ensure that runtime reconfiguration requests can be made only from the local host. Further, you would like the host to synchronize with only one of a pair of offsite servers or, failing that, a time source on the class B network whose address is 128.212. The following entries in the configuration file would implement this policy:

```
# By default, do not trust and do not allow modifications
restrict default notrust nomodify

# These hosts are trusted for time, but no modifications allowed
restrict 128.212.0.0 mask 255.255.0.0 nomodify
restrict 128.192.8.4 nomodify
restrict 192.211.1.24 nomodify

# These local addresses are unrestricted
restrict 128.212.66.67
restrict 127.0.0.1
```

The first entry is the default entry, which all hosts match and hence which provides the default set of flags. The next three entries indicate that matching hosts have only the **nomodify** flag set and hence are trusted for time. If the mask is not specified in the **restrict** statement, it defaults to 255.255.255.255.

Note that the address 128.212.66.67 matches three entries in the table, the default entry (mask 0.0.0.0), the entry for net 128.212 (mask 255.255.0.0) and the entry for the host itself (mask 255.255.255.255). As expected, the flags for the host are derived from the last entry, as that mask has the most bits set.

Each **restrict** statement applies to packets from all hosts, including those that are configured elsewhere in the configuration file. Hence, if you specify a default set of restrictions that you do not wish to apply to the hosts you are synchronizing with, you must override the default restrictions for those hosts with additional **restrict** statements.

Name resolution

The **xntpd** daemon can specify host names requiring resolution in **peer** and **server** statements in the configuration file. This task is actually accomplished by a separate program, **xntpres**. When the daemon comes across a **peer** or **server** statement in the configuration file where the host is identified by name rather than by IP address, it records the relevant information and continues. When the end of the configuration file has been reached and one or more entries requiring name resolution have been found, **xntpres** is started. The **xntpd** daemon continues running at the same time to process those hosts with numeric addresses.

The **xntpres** program attempts to resolve each host name, meaning that it tries to obtain the IP address associated with the named host. If it succeeds in resolving the name, it reconfigures the local host to add the newly resolved host to the list of time servers to be polled. The runtime reconfiguration of the local host is accomplished using the same mode 7 runtime reconfiguration facility that **xntpdc** uses. If **xntpres** is at first unable to resolve a host's name, it retries periodically until it succeeds or until it determines that the name cannot be resolved.

There are several configuration requirements if **xntpres** is to be used. The pathname of the **xntpres** program must be made known to the daemon with the **resolver** statement in the configuration file. Also, mode 7 runtime reconfiguration must be enabled with the **requestkey** statement and a list of valid keys (valid keys are specified in the *keys* file).

The following fragment might be added to */etc/ntp.conf* to accomplish this:

```
resolver /local/etc/xntpres
keys /etc/ntp.keys
requestkey 65535
```

Note that **xntpres** sends packets to the server with a source address of 127.0.0.1. If you are using the address-and-mask facility in the configuration file, you must take care not to restrict modification requests from this address or **xntpres** fails.

Sample scenarios

This first scenario would be used when configuring a single host to use NTP. The host *rainbow.sci.com* is expected to run as a stratum 2 server because it is configured to get the time from two time servers with radio clocks, *bedivere.cs.purdue.edu* and *asmus1.genetics.uga.edu*. The **xntpd** daemon stores the frequency error of the clock on *rainbow.sci.com* in */etc/driftfile*. This is a good example to use if you just want to get **xntpd** up and running quickly. Note that it can take a day or two for a host to synchronize with a time server.

Here is the *ntp.conf* file for *rainbow.sci.com*.

```
#
# Peer configuration for 128.212.66.13 (rainbow.sci.com)
#
peer 128.211.1.24       # bedivere.cs.purdue.edu
peer 128.192.8.4        # asmus1.genetics.uga.edu

driftfile /etc/driftfile
```

This next scenario is the complete configuration for *sci.com*. The configuration files here configure *sci.com* to match the topology discussed in "Guidelines" earlier in this chapter. All hosts can be configured dynamically with **xntpd**. There are no restrictions on which hosts can be designated as time servers in a given host's configuration file.

Here is the *ntp.conf* file for *Zite*.

```
#
# Peer configuration file for 128.212.66.90 (zite.sci.com)
#
peer 128.211.1.4      # percival.cs.purdue.edu
peer 192.35.53.67     # deidinator.psyc.chi.il.us
peer HGWells.Sci.com
peer DrWho.Sci.com

broadcast 128.212.66.255

driftfile  /etc/ntp.drift
resolver   /etc/xntpres
keys       /etc/ntp.keys
requestkey  65534
controlkey  65535
```

Here is the *ntp.keys* file for *Zite*.

```
2313      A    APassword
65534     A    NoSecret
65535     A    BadKey
```

Here is the *ntp.conf* file for *HGWells*.

```
#
# Peer configuration for HGWells (128.212.66.67)
#
peer 128.192.8.4    # asmus1.genetics.uga.edu
peer 128.211.1.24   # bedivere.cs.purdue.edu
peer zite
peer DrWho

broadcast 128.212.66.255

driftfile  /etc/ntp.drift
resolver   /etc/xntpres
keys       /etc/ntp.keys
requestkey  65534
controlkey  65535
```

Here is the *ntp.keys* file for *HGWells*.

```
2313        A     APassword
65534       A     NoSecret
65535       A     BadKey
```

Here is the *ntp.conf* file for *DrWho*.

```
#
# Peer configuration file for 128.212.66.118
#                              DrWho.Sci.com
#
peer 192.35.54.2       # nehd.gov
peer 192.35.55.2       # berdie.athens.ga.us
peer HGWells.Sci.com
peer Zite.Sci.com

broadcast 128.212.66.255

driftfile  /etc/ntp.drift
resolver   /etc/xntpres
keys       /etc/ntp.keys
requestkey  65534
controlkey  65535
```

Here is the *ntp.keys* file for *DrWho*.

```
2313        A     APassword
65534       A     NoSecret
65535       A     BadKey
```

Here is the *ntp.conf* file for *Moondoggie.*

```
# Peer configuration used by all stratum 3
#      servers at Sci.com
#
broadcastclient  yes
broadcastdelay   0.0500
driftfile        /etc/ntp.drift
resolver         /etc/xntpres
keys             /etc/ntp.keys
requestkey  65534
controlkey  65535
```

Here is the *ntp.keys* file for *Moondoggie.*

```
2313      A    APassword
65534     A    NoSecret
65535     A    BadKey
```

Testing and tuning

There are several parameters available for tuning time servers. These are set using the **precision** and **maxskew** configuration statements. A fragment that would simply reset these to their default values follows:

```
precision    -6
maxskew      0.01
```

The **precision** statement sets the value of **sys.precision**, and the **maxskew** statement sets the value of **NTP.MAXSKW**. The variable **sys.precision** is the base two logarithm of the expected precision of the system clock. The default value is -6 on all servers.

The NTP protocol makes use of **sys.precision** in several places. For one, **sys.precision** is included in packets sent to peers and is used by them as a kind of quality indicator. When faced with selecting for synchronization purposes one of several servers, all of which are at the same stratum and offer about the same network path delay, clients prefer to synchronize to those claiming the smallest (most negative) value of **sys.precision**. The effect is particularly pronounced when all the servers are on the same LAN. Hence, if you run several stratum 1 servers, or three or four stratum 2 servers, and you would like clients to prefer one of these over the others for synchronization, you can accomplish this by decreasing the value of **sys.precision** on the preferred server or by increasing this value on the other servers, or by doing both.

The other tuning parameter is the **antihop aperture** and is derived from **sys.precision** and **NTP.MAXSKW** using the following equation:

```
antihop aperture = 2 ** sys.precision + NTP.MAXSKW
```

This equation says that the antihop aperture is equal to 2 raised to the **sys.precision** power plus **NTP.MAXSKW.**

Making the **antihop aperture** larger makes it less likely that the host will "hop" from the server it is currently synchronizing with to a different server. Unfortunately, this also increases the probability that the host continues to synchronize with a server whose clock is no longer accurate.

Making the **antihop aperture** smaller allows the host to hop more freely from server to server, but this can also cause it to generate a fair bit more NTP packet traffic than necessary and to no good purpose. Given the agreement among current stratum 1 NTP servers and the performance typical of the Internet, it is recommended that the **antihop aperture** be kept between 0.020 and 0.030. The default value is about 0.026.

You can change the **antihop** aperture by changing the value of **NTP.MAXSKW** using the **maxskew** configuration statement. Note, however, that if you wish to change **sys.precision** with the **precision** configuration statement, but do not wish to alter the **antihop aperture**, you must change **NTP.MAXSKW** to compensate.

Query commands

Two query programs, **ntpq**(ADMN) and **xntpdc**(ADMN), are available for use by the network administrator. The **ntpq** command sends queries and receives responses using NTP standard mode 6 control messages, while the **xntpdc** command uses NTP private mode 7 messages to send its requests. The format and content of these messages are specific to **xntpdc**. This command allows you to inspect a wide variety of internal counters and other state data. It also provides you with an interface to the runtime reconfiguration facility.

Both **xntpdc** and **xntpd** use seconds as the unit of time, both when reading the configuration file and when doing runtime reconfiguration. Both programs also print values in seconds. On the other hand, **ntpq** prints all time values in milliseconds. You must, therefore, take special care to convert values output by **ntpq** from milliseconds to seconds before modifying the configuration file, either manually or with **xntpdc**.

The **xntpd** daemon is designed to allow its configuration to be modified at runtime. Nearly everything that can be configured in the configuration file *ntp.conf* may be altered using **xntpdc**.

Mode 6 and mode 7 packets, which would modify the configuration of the local host, must be authenticated. To enable the facilities one must, in addition to specifying the location of a *keys* file, indicate in the configuration file the key IDs to be used for authenticating reconfiguration requests. The following fragment might be added to the configuration file to enable **ntpq** and **xntpdc**.

```
keys /etc/ntp.keys
requestkey 65535        # for mode 7 requests (used by xntpdc)
controlkey 65534        # for mode 6 requests (used by ntpq)
```

If the **requestkey** or the **controlkey** configuration statement is omitted from the configuration file, the corresponding runtime reconfiguration facility is disabled.

The query programs require the user to specify a key ID and a key to use for authenticating requests to be sent. The key ID provided must be the same one specified in the configuration file, while the key value provided must match the one corresponding to the given key ID in the *keys* file.

Further examples

This example demonstrates the use of **ntpq** and **xntpdc** to list the servers that *Moondoggie.sci.com* is currently polling. The server marked with an asterisk "*" is the one that *Moondoggie* is currently synchronizing with. *Moondoggie* is also listening for broadcast packets on the subnet with IP address 128.212.66, but no packets are currently being broadcast there. Note the difference in the units used by **ntpq** and **xntpdc** to display delay, offset, and dispersion. The former uses milliseconds, whereas the latter uses seconds.

```
sjc@moondoggie $ ntpq
ntpq> peer

     remote           refid      st  when poll reach delay offset   disp
==============================================================================
 128.212.66.255  0.0.0.0        16 never   64     0    0.0   0.00  64000
*Zite.sci.com    deidinator.ps   2    53   64   376   81.1  -1.00   12.4
+DrWho.sci.com   berdie.athens   2    11   64    37   41.2  30.00 7504.3
```

```
sjc@moondoggie $ xntpd
xntpd> peer

     remote           local      st poll reach  delay   offset   disp
==============================================================================
*Zite.sci.com    192.35.53.67     2   64  376  0.0811 -0.001003 0.0124
-DrWho.sci.com   192.35.55.2      2   64   37  0.0412  0.030001 7.5043
^128.212.66.255  0.0.0.0         16   64    0  0.0000  0.000000 64.000
```

This information tells us, among other things, that:

- *Moondoggie* is currently synchronizing with *Zite*.
- *Zite* is at stratum 2, implying that *Moondoggie* is at stratum 3.
- *Moondoggie* is polling *Zite* once every 64 seconds.
- The reachability register for *Zite* is octal 376.
- The clock on *Moondoggie* is 1 millisecond, or more precisely, 1003 microseconds, behind the clock on *Zite*.
- The roundtrip delay to *Zite* is 81 milliseconds.
- The dispersion is 12.4 milliseconds.

The next example shows how to turn on monitoring and how to list the statistics that are gathered when monitoring is enabled.

```
xntpdc> monitor on
Keyid: 65534
Password: NoSecret
done!
```

Now that monitoring is on, we can let the daemon run for a while, then display statistics, as shown here.

```
xntpdc> monlist

   address       port     count  mode version lasttime firsttime
==============================================================================
127.0.0.1        1417         6    7     2         0    475668
128.212.66.118    123      3818    1     2        92    475332
128.212.66.90     123      7432    5     2        48    475632
```

This display tells us who has connected to the **xntpd** daemon, on what UDP port the connection was made, the number of packets received, the mode of this connection, the version of NTP being run, the time since the last packet exchange in seconds, and the time since the first packet exchange in seconds.

The last example shows how we can dynamically delete and add time servers. First, we indicate which host we wish to communicate with, then we display the current peers of that host.

```
xntpdc> host rainbow
current host set to rainbow.sci.com

xntpdc> peers

     remote           local  st poll reach  delay   offset   disp
==============================================================================
*zite.sci.com    128.211.1.4  2   64  376  0.0512 -0.010000 0.0021
+HGWells.sci.com 0.0.0.0     16 1024    0  0.0000  0.000000 64.000
```

Now we remove *HGWells* as a peer and view the result.

```
xntpdc> unconfig HGWells
Keyid: 65534
Password: NoSecret
done!

xntpdc> peers

     remote        local  st poll reach  delay   offset   disp
==============================================================================
*zite.sci.com 128.211.1.4  2   64  376  0.0512 -0.010000 0.0021
```

Now that *HGWells* has been removed as a peer, we go ahead and add host *DrWho* as a new peer.

```
xntpdc> addpeer DrWho.sci.com
done!

xntpdc> peers

     remote           local  st poll reach  delay   offset   disp
==============================================================
+DrWho.sci.com 0.0.0.0     16   64     0  0.0000  0.000000 64.000
*zite.sci.com  128.211.1.4  2   64   376  0.0512 -0.010000 0.0021
```

If we allow *Rainbow* to poll the new peer for a period of time and then reissue the **peers** subcommand, we might see something like this display.

```
xntpdc> peers

     remote           local      st poll reach  delay   offset   disp
==============================================================
*DrWho.sci.com 128.212.66.118  2   64  377  0.0312 -0.004488 0.0011
~zite.sci.com  128.211.1.4     2   64  376  0.0512 -0.010000 0.0021
```

Troubleshooting

The **xntpd** daemon records all problems and errors using the **syslog**(SLIB) system call into the file */usr/adm/syslog*. You can examine this log for signs of trouble. For example, too many entries indicating that the local clock was stepped instead of slewed to the correct time suggests that the daemon is having trouble synchronizing the clock. (In this instance, more than two entries per hour on a consistent basis is probably too many.) The most likely cause is that some other program, perhaps **timed**, is also setting or slewing the clock.

Another sign of more than one entity setting or slewing the clock is the following message in the *syslog* file:

```
Previous time adjustment did not complete
```

One of the following messages is issued when the clock of another host (the remote clock) is off by more than 1000 seconds from the clock on the local host (the local clock):

```
Clock appears to be number_of_seconds seconds fast,
something may be wrong
```

```
Clock appears to be number_of_seconds seconds slow,
something may be wrong
```

In this situation **xntpd** will not try to synchronize with that host. If you want to synchronize with that host, you will need to set the local clock to within 1000 seconds of the remote clock. You can accomplish this by using the **ntpdate**(ADMN) command.

Running mixed synchronization subnets

The **xntpd** time daemon is an implementation of NTP Version 2. By default, **xntpd** sends NTP packets marked as being of the Version 2 variety. However, another implementation of the NTP time daemon, called **ntpd**, may be running on hosts you wish to poll. If this is the case, you must include the **version** option in the **peer** or **server** configuration statements for those hosts to force **xntpd** to send packets of the Version 1 variety.

For example, if the host *nehd.gov* at IP address 192.35.54.2 is a host you wish to poll in symmetric active mode and you know that *nehd.gov* is running **ntpd** instead of **xntpd**, you should include the following configuration statement in your local hosts *ntp.conf* file:

```
peer 192.35.54.2 version 1      # nehd.gov (running ntpd)
```

This statement ensures that Version 1 NTP packets are sent by your local host to *nehd.gov*.

Chapter 14

TCP/IP sendmail administration

The **sendmail**(ADMN) utility provides a "back-end" mail interface to other mail programs; that is, **sendmail** only routes and delivers mail, it does not provide a user interface for composing or formatting letters. **sendmail** provides network access and supports either Internet-style addressing, such as user@domain or UUCP-style addressing, such as host!user. **sendmail** works with delivery agents such as SMTP (Simple Mail Transfer Protocol), X.400, and UUCP. **sendmail** only handles text format files; binary data is not allowed.

NOTE This chapter describes how you can manually fine-tune **sendmail** configuration. In most installations, running the initialization script **mkdev sendmail-init** without any manual configuration will produce the desired results on your system. In that case, the information found in this chapter following the section "Configuring a standard installation" is unnecessary, and is provided for completeness only.

You do not need to use **sendmail** to route mail; the operating system's MMDF mail router is also available and provides several key benefits over using **sendmail**, including an easier configuration process. For more information on MMDF, refer to your operating system's .

This chapter compares **sendmail** to other mailers, discusses how **sendmail** works, explains **sendmail** configuration, shows how to run **sendmail**, and details **sendmail** administration.

sendmail and other mailers

This section compares **sendmail** with three other mail programs:

- **delivermail**
- MMDF (Multichannel Memorandum Distribution Facility)
- MPM (Message Processing Module)

These comparisons are provided for those who are familiar with these other mail routing programs.

Comparing sendmail with delivermail

The **sendmail** program is an outgrowth of **delivermail**. The primary differences are as follows:

- Configuration information is not compiled in. This change simplifies many of the problems of moving to other machines. It also simplifies debugging of new mailers.
- Address parsing is more flexible. For example, **delivermail** only supported one gateway to any network, whereas **sendmail** can be sensitive to host names and reroute to different gateways.
- The forward and include features of **sendmail** eliminate the requirement that the system alias file be writable by any user (or that an update program be written, or that the system administration make all changes).
- The **sendmail** program supports message batching across networks when a message is being sent to multiple recipients.
- A mail queue is provided in **sendmail**. Mail that cannot be delivered immediately but can potentially be delivered later is stored in this queue for a later retry. The queue also provides a buffer against system crashes; after the message has been collected, it may be reliably redelivered even if the system crashes during the initial delivery.
- The **sendmail** program uses the networking support of the operating system to provide a direct interface to networks such as Ethernet using SMTP (the Simple Mail Transfer Protocol) over a TCP/IP connection.

Comparing sendmail with MMDF

The Multichannel Memorandum Distribution Facility (MMDF) spans a wider problem set than **sendmail**. For example, the domain of MMDF includes a "phone network" mailer, whereas **sendmail** calls on pre-existing mailers in most cases.

MMDF and **sendmail** both support aliasing, customized mailers, message batching, automatic forwarding to gateways, queueing, and retransmission. MMDF supports two-stage timeout, which **sendmail** does not support. MMDF uses two-stage timeout when routing mail through machines to users. If a message cannot be forwarded to a particular machine or to a particular user on a machine, a warning is sent back to the mail message sender. This is stage 1. At some future time (configurable by the administrator), the message is relayed again. If it fails, a failure message is returned to the sender, and MMDF makes no further attempts to resend the original message. This is stage 2.

MMDF does offer several substantial benefits over **sendmail**, including:

- Configuration files that are easy to read and understand.
- The ability for end-users to configure their own sorting parameters.
- A larger set of supported delivery agents.

Because MMDF does not consider backwards compatibility as a design goal, the address parsing is simpler but much less flexible.

It is somewhat harder to integrate a new channel into MMDF. In particular, MMDF must know the location and format of host tables for all channels, and each channel must speak a special protocol. This allows MMDF to do additional verification (such as verifying host names) at submission time.

MMDF strictly separates the submission and delivery phases. Although **sendmail** has the concept of each of these stages, they are integrated into one program, whereas in MMDF they are split into two programs.

Sendmail and the message-processing module (MPM)

The Message Processing Module (MPM) matches **sendmail** closely in terms of its basic architecture. However, like MMDF, the MPM includes the network interface software as part of its domain.

MPM also postulates a duplex channel to the receiver, as does MMDF. This allows simpler handling of errors by the mailer than is possible in **sendmail**. When a message queued by **sendmail** is sent, any errors must be returned to the sender by the mailer itself. Both MPM and MMDF mailers can return an immediate error response, and a single error processor can create an appropriate response.

MPM prefers passing the message as a structured object, with {type, length, value} triples. Such a convention requires a much higher degree of cooperation between mailers than is required by **sendmail**. MPM also assumes a universally agreed-upon Internet name space (with each address in the form of a net-host-user tuple), which **sendmail** does not.

How sendmail works

When a sender, for example the **mail**(C) command, wants to send a message, a request is issued to **sendmail** directly, via SMTP over a pair of pipes, or via SMTP using a socket. **sendmail** then collects and queues the message along with other messages, and when ready delivers the message. If any errors occur while delivering the message, **sendmail** returns a message to the sender describing the error. Alternatively, it may return a status code to indicate what went wrong.

If **sendmail** is called using SMTP over pipes or through a socket, the arguments are first scanned and option specifications are processed. Recipient addresses are then collected, either from the command line or from the SMTP command RCPT (recipient), and a list of recipients is created. Aliases are expanded at this point. This includes any aliases that are part of a mailing list. At this stage, as much validation of the addresses as possible is done. Syntax is checked and local addresses are verified, but detailed checking of host names and addresses is deferred until delivery. Forwarding is also performed as the local addresses are verified.

The **sendmail** program appends each address to the recipient list after parsing the recipient list. When a name is aliased or forwarded, the old name is retained in the list, and a flag is set that tells the delivery phase to ignore this recipient. This list is kept free from duplicates, preventing alias loops and duplicate messages from being delivered to the same recipient, as might occur if a person is in two groups.

Collecting messages

The **sendmail** program collects the message after the address parsing is complete. The message should have a header at the beginning. No formatting requirements are imposed on the message, except that they must be lines of text; binary data is not allowed. A message consists of a header and a body, which are separated by a blank line. The header is parsed and stored in memory, and the body of the message is saved in a temporary file.

To simplify the program interface, the message is collected even if no addresses are valid. The message is then returned to the sender with an error.

The header is formatted as a series of lines of the form:

field-name: field-value

field-value can be split across lines by starting the lines that follow with a space or a tab. Some header fields have special internal meaning, in which

case they are subject to special processing. Other headers are simply passed through. Some header fields, such as time stamps, may be added automatically.

The message body is a series of text lines. It is completely uninterpreted and untouched by **sendmail**, except that lines beginning with a dot "." have the dot doubled when transmitted over an SMTP channel. This extra dot is stripped by the receiver. (SMTP uses lines beginning with a dot to signal the end of the message.)

Delivering messages

For each unique mailer and host in the recipient list, **sendmail** calls the appropriate mailer. Each mailer invocation sends to all users receiving the message on one host. Mailers that accept only one recipient at a time are handled accordingly.

The message is sent to the mailer, using one of the same three interfaces used to submit a message to **sendmail**. Each copy of the message is prefaced by a customized header. The mailer status code is caught and checked, with a suitable error message given as appropriate. The exit code must conform to a system standard; otherwise, a generic message such as `Service unavailable` is given.

The send queue is sequenced by the receiving host before transmission in order to implement message batching. Each address is marked as it is sent, and so rescanning the list is safe. An argument list is built as the scan proceeds. Mail to files is detected during the scan of the send list.

After a connection is established, **sendmail** makes the changes to the header necessary for correct interpretation by a particular mailer and sends the result to that mailer. If any mail is rejected by the mailer, a flag is set to invoke the return-to-sender function after all delivery completes.

Queueing for retransmission

If the mailer returns a status code indicating that the message should be sent later, **sendmail** puts the mail on the queue and tries again later.

Return to sender

If errors occur during processing, **sendmail** returns the message to the sender for retransmission. If the user agent (mail) detects the error, then it is put in the *dead.letter* file located in the sender's home directory. If a **sendmail** server is connecting with a **sendmail** client on another machine, then the user is presumed to have become detached from the transaction, and so the message is mailed back to them.

Editing the message header

A certain amount of editing occurs automatically to the message header. Header lines can be inserted under control of the configuration file. Some lines can be merged. For example, a "From:" line and a "Full-name:" line can be merged under certain circumstances.

Aliasing, forwarding and including mail

The **sendmail** program reroutes mail in any of the following three ways:

- by aliasing
- by forwarding
- by inclusion

Aliasing applies across the entire system. Forwarding allows all users to reroute incoming mail destined for their accounts. Inclusion directs **sendmail** to read a file for a list of addresses. Forwarding is normally used in conjunction with aliasing. Each of these methods is described in more detail in the next three sections.

Aliasing

Aliasing matches names to address lists using a system-wide file. This file is indexed to speed access. Only names that parse as local are allowed as aliases. This guarantees a unique key. The alias file is usually configured to be */usr/lib/aliases*. This file is not in the same format as the alias file */usr/lib/mail/aliases*. The identity of the alias file is configured through the *sendmail.cf* file.

Forwarding

After aliasing, recipients that are local and valid are checked for the existence of a *.forward* file in their home directory. If one exists, the message is not sent to that user, but rather to the list of users in that file. Often, this list contains a single address, and is used only for network mail forwarding.

Forwarding also permits a user to specify a private incoming mailer. For example, this forwarding uses a different incoming mailer:

```
"|/usr/local/newmail myname"
```

Including

The syntax for including a file is:

```
:include: pathname
```

An address of this form reads the file specified by ***pathname*** and sends to all users listed in that file.

The intention is not to support direct use of this feature, but rather to use this as a subset of aliasing. In the following example, the form of the alias used is a method of letting a project maintain a mailing list without interaction with the system administration, even if the alias file is protected.

```
project: :include:/usr/project/userlist
```

It is not necessary to rebuild the index on the alias database when a list of this type is changed. All that is needed is to edit the include file to reflect the changes. In this example, the include file is */usr/project/userlist*.

Queued messages

If the mailer returns a `temporary failure` exit status, the message is queued. A control file describes the recipients to be sent to and various other parameters. This control file is formatted as a series of lines, each describing a sender, a recipient, the time of submission, or some other significant parameter of the message. The header of the message is stored in the control file, so that the associated data file in the queue is just the temporary file that was originally collected.

Configuring sendmail

There are three basic steps to installing **sendmail.** They are:

- installing the software, as described in the *SCO TCP/IP Release and Installation Notes*
- running **mkdev sendmail-init** to create the **sendmail** configuration file
- editing the **sendmail** configuration file to fine-tune **sendmail** operation

Because the basic configuration file is adequate for most sites, the third step is often unnecessary.

The configuration file is read by **sendmail** reads when the program starts. This file describes the mailers that **sendmail** knows about, how to parse addresses, how to rewrite the message header, and the settings of various options.

Configuring a standard installation

To use the **sendmail** mail system, you must first run **mkdev sendmail-init,** which initializes **sendmail** and configures it by invoking **mkdev cf.** After first initializing **sendmail,** you can reconfigure it at any time by running **mkdev cf.**

The **mkdev cf** script supports both uucp and Internet style addressing. The uucp configuration allows a network administrator to designate a machine on the network as the UUCP Gateway Machine, which will process all UUCP requests for the net or subnet.

To configure a standard installation with **mkdev sendmail-init**:

1. Type **mkdev sendmail-init** at the command line. The message `Saving the following files:` appears, followed by a list of files and this message: `The mail system will now use sendmail for delivering messages.`

   The **mkdev sendmail-init** command installs and configures **sendmail**. The **mkdev cf** command reconfigures **sendmail**.
2. A list of files is displayed, along with the message: `Press ⟨Return⟩ to continue.`
3. A menu appears. You must go through the various options of the menu in sequence.
4. The first menu option is to edit UUCP connections. Default values can be taken from the system, but you need to know if your machine is a UUCP gateway or not.
5. The second menu option is to edit the domain name. The default domain name is taken from the **hostname** utility; otherwise, you can specify a domain name.
6. The third menu option is to edit alternate host names. This menu selection is optional.
7. The fourth menu option is to edit network configuration information. The default answers are all "no" for the questions displayed in this option. Change the answers if necessary. If your machine is a UUCP mail gateway, then answer "yes" to the question `Is there a UUCP gateway?`.
8. The fifth menu option is simply a display that allows you to review your answers from the preceding options. If necessary, go back and correct any errors at this point by returning to the earlier option. This menu selection is optional.
9. The sixth menu option generates the new *sendmail.cf* file. A display of your selections is presented first, then you can generate the new file.

   You are prompted with this message: `Do you wish to change anything? [y/n]` The old *sendmail.cf* file is saved as */usr/lib/custom/save/sendmail.cf*.
10. The seventh menu option lets you quit the **sendmail** configuration menu.

Configuring a non-standard installation

If the default configuration performed by **mkdev sendmail-init** is not adequate for your system, you will need to edit the **sendmail** configuration file, *sendmail.cf*, then run **mkdev cf** to effect the changes you made. This section describes the configuration file in detail, including hints on how to write one of your own if you have to.

There is one point that should be made clear immediately: the syntax of the configuration file is designed to be reasonably easy to parse, because this is done every time **sendmail** starts up. As a result, the configuration file is not particularly easy for a human to read or write.

An overview of the configuration file is given first, followed by details of the semantics.

The syntax

The configuration file is organized as a series of lines, each of which begins with a single character defining the semantics for the rest of the line. Lines beginning with a space or a tab are continuation lines (although the semantics are not well defined in many places). Blank lines and lines beginning with a number sign "#" are comments.

R and S - rewriting rules

The core of address parsing is the rewriting rules. These are an ordered production system. The **sendmail** command scans through the set of rewriting rules looking for a match on the left-hand side (***lhs***) of the rule. When a rule matches, the address is replaced by the right-hand side (***rhs***) of the rule.

There are several sets of rewriting rules. Some of the rewriting sets are used internally and must have specific semantics. Other rewriting sets do not have specifically assigned semantics, and may be referenced by the mailer definitions or by other rewriting sets.

The syntax of these two commands is:

S*n* — Sets the current ruleset being collected to ***n***. If you begin a ruleset more than once, it deletes the old definition.

R*lhs rhs comments* — The ***lhs*** is a pattern that is applied to the input. If it matches, the input is rewritten to the ***rhs***. The ***comments*** are ignored. The fields must be separated by at least one tab character; there may be embedded spaces in the fields.

D - define macro

Macros are named with a single character. These can be selected from the entire ASCII set, but user-defined macros should be selected from the set of uppercase letters only. Lowercase letters and special symbols are used internally.

The syntax for macro definitions is:

D*x val*

Here ***x*** is the name of the macro and ***val*** is the value it should have. Macros can be interpolated in most places using the escape sequence **$***x*.

C and F - define classes

Classes of words may be defined to match on the left-hand side of rewriting rules. For example, a class of all local names for this site might be created so that attempts to send to oneself can be eliminated. These can either be defined directly in the configuration file or read in from another file. Classes may be given names from the set of uppercase letters. Lowercase letters and special characters are reserved for system use.

The syntax is:

C*c word1 word2...*
F*c file*

The first form defines the class *c* to match any of the named words. It is permissible to split them among multiple lines, for example:

CHmonet ucbmonet

This example is equivalent to the following:

CHmonet
CHucbmonet

The second form reads the elements of the class *c* from the named *file*.

M - define mailer

Programs and interfaces to mailers are defined in this line. The format is:

M*name, {field=value }**

Here ***name*** is the name of the mailer (used internally only) and the *"field=name"* pairs define attributes of the mailer. Fields are:

Path	pathname of the mailer
Flags	special flags for this mailer
Sender	rewriting set for sender addresses
Recipient	rewriting set for recipient addresses
Argv	argument vector to pass to this mailer
Eol	end-of-line string for this mailer
Maxsize	The maximum message length for this mailer

Only the first character of the field name is checked.

H - define header

The format of the header lines that **sendmail** inserts into the message is defined by the **H** line. The syntax of this line is:

H[*?mflags?*]*hname: htemplate*

Continuation lines in this specification are reflected directly into the outgoing message. The ***htemplate*** is macro-expanded before insertion into the message. If the ***mflags*** (surrounded by question marks) are specified, at least one of the specified flags must be stated in the mailer definition for this header to be automatically output. If one of these headers is in the input, it is reflected to the output, regardless of these flags.

Some headers have special semantics, described in the following sections.

O - set option

There are a number of "random" options that can be set from a configuration file. Options are represented by single characters. The syntax of this line is:

O*o value*

This sets option *o* to be ***value***. Depending on the option, ***value*** can be a string, an integer, a Boolean (with legal values "t", "T", "f", or "F"; the default is TRUE), or a time interval.

T - define trusted users

Trusted users are those who are permitted to override the sender address using the **-f** flag. These typically are *root, uucp,* and *network,* but in some cases it may be convenient to extend this list to include other users, perhaps to support a separate UUCP login for each host. The syntax of this line is:

T*user1 user2...*

There may be more than one of these lines.

P - precedence definitions

Values for the "Precedence:" field may be defined using the **P** control line. The syntax of this field is:

P*name=num*

When the ***name*** is found in a "Precedence:" field, the message class is set to ***num***. Higher numbers mean higher precedence. Numbers below zero have the special property that error messages cannot be returned. The default precedence is zero. For example, our list of precedences is:

```
Pfirst-class=0
Pspecial-delivery=100
Pjunk=-100
```

The semantics

This section describes the semantics of the configuration file.

Special macros, conditionals

Macros are interpolated using the construct **$*x***, where *x* is the name of the macro to be interpolated. In particular, lowercase letters are reserved to have special semantics, used to pass information in or out of **sendmail**; some special characters are reserved to provide conditionals; and so on.

Conditionals can be specified using the syntax:

$?x ***text1*** **$|** ***text2*** **$.**

This interpolates ***text1*** if the macro **$x** is set, and ***text2*** otherwise. The "else" (**$|**) clause can be omitted.

The following macros must be defined to transmit information into **sendmail**:

- **e** SMTP entry message
- **j** "official" domain name for this site
- **l** format of the UNIX system from line
- **n** name of the daemon (for error messages)
- **o** set of "operators" in addresses
- **q** default format of sender address

The **$e** macro is printed out when SMTP starts up. The first word must be the **$j** macro. The **$j** macro should be in RFC821 format. The **$l** and **$n** macros can be considered constants, except under terribly unusual circumstances. The **$o** macro consists of a list of characters that are considered tokens and that separate tokens when parsing. For example, if "**@**" were in the **$o** macro, then the input **a@b** would be scanned as three tokens: "a", "**@**", and "b". Finally, the **$q** macro specifies how an address should appear in a message when it is defaulted. For example, on our system, these definitions are:

```
De$j Sendmail $v ready at $b
DnMAILER-DAEMON
DlFrom $g  $d
Do.:%@!^=/
Dq$g$?x ($x)$.
Dj$H.$D
```

An acceptable alternative for the **$q** macro is $?x$x $.<$g>. These correspond to the following two formats:

```
eric@Ocelot (Eric Allman)
Eric Allman <eric@Ocelot>
```

Some macros are defined by **sendmail** for interpolation into argv's for mailers or for other contexts. These macros are:

a	origination date in Arpanet format
b	current date in Arpanet format
c	hop count
d	date in UNIX (ctime) format
f	sender (from) address
g	sender address relative to the recipient
h	recipient host
i	queue id
p	**sendmail**'s pid
r	protocol used
s	sender's host name
t	numeric representation of the current time
u	recipient user
v	version number of **sendmail**
w	hostname of this site
x	full name of the sender
z	home directory of the recipient

There are three types of dates that can be used. The **$a** and **$b** macros are in Arpanet format; **$a** is the time as extracted from the "Date:" line of the message (if there was one), and **$b** is the current date and time (used for postmarks). If no "Date:" line is found in the incoming message, **$a** is set to the current time also. The **$d** macro is equivalent to the **$a** macro in UNIX system (ctime) format.

The **$f** macro is the ID of the sender as originally determined; when you are mailing to a specific host, the **$g** macro is set to the address of the sender *relative to the recipient*. For example, if you send to *bollard@matisse* from the machine *ucbarpa*, the **$f** macro is "eric" and the **$g** macro is "eric@ucbarpa."

The **$x** macro is set to the full name of the sender. This can be determined in several ways. It can be passed as a flag to **sendmail.** The second choice is the value of the "Full-name:" line in the header if it exists, and the third choice is the comment field of a "From:" line. If all of these fail, and if the message is being originated locally, the full name is looked up in the */etc/passwd* file.

When sending, the **$h, $u,** and **$z** macros are set to the host, user, and home directories (if local) of the recipient. The first two are set from the **$@** and **$:** parts of the rewriting rules, respectively.

The **$p** and **$t** macros create unique strings (for example, for the "Message-Id:" field). The **$i** macro is set to the queue ID on this host; if put into the timestamp line, it can be extremely useful for tracking messages. The **$v** macro is set to be the version number of **sendmail;** this is normally put in

timestamps and has proved extremely useful for debugging. The **$w** macro is set to the name of this host, if it can be determined. The **$c** field is set to the "hop count," that is, the number of times this message has been processed. This can be determined by the **-h** flag on the command line or by counting the timestamps in the message.

The **$r** and **$s** fields are set to the protocol used to communicate with **sendmail** and the sending ***hostname***; these are not supported in the current version.

Special classes

Set the class **$=w** to be the set of all names by which this host is known. This can be used to delete local hostnames.

The left-hand side

The left-hand side of rewriting rules contains a pattern. Normal words are simply matched directly. Metasyntax is introduced using a dollar sign. The metasymbols are:

- **$*** match zero or more tokens
- **$+** match one or more tokens
- **$-** match exactly one token
- **$=***x* match any token in class *x*
- **$~***x* match any token not in class *x*

If any of these match, they are assigned to the symbol **$***n* for replacement on the right-hand side, where ***n*** is the index in the LHS. For example:

```
$-:$+
```

Suppose the LHS is applied to the following input:

```
UCBARPA:eric
```

The rule matches, and the values passed to the RHS are:

```
$1  UCBARPA
$2  eric
```

The right-hand side

When the left-hand side of a rewriting rule matches, the input is deleted and replaced by the right-hand side. Tokens are copied directly from the RHS, unless they begin with a dollar sign. Metasymbols are:

$*n*	substitutes indefinite token ***n*** from LHS
$[***name***$]	canonicalizes ***name***
$>*n*	calls ruleset ***n***
$#*mailer*	resolves to ***mailer***
$@*host*	specifies ***host***
$:*user*	specifies ***user***

The **$*n*** syntax substitutes the corresponding value from a **$+**, **$-**, **$***, **$=**, or **$~** match on the LHS. It can be used anywhere.

A host name enclosed between **$[** and **$]** is looked up using the **gethostent**(3) routines and replaced by the canonical name. For example, $[[128.32.130.2]$] would become vangogh.berkeley.edu, and $[csam$] would become lbl-csam.arpa.

The **$>*n*** syntax causes the remainder of the line to be substituted as usual and then passed as the argument to ruleset ***n***. The final value of ruleset ***n*** then becomes the substitution for this rule.

The **$#** syntax should only be used in ruleset zero. It causes evaluation of the ruleset to terminate immediately, and it signals to **sendmail** that the address has completely resolved. The complete syntax is:

$#*mailer*$@*host*$:*user*

This specifies the {mailer, host, user} 3-tuple necessary to direct the mailer. If the mailer is local, the host part can be omitted. The ***mailer*** and ***host*** must be a single word, but the ***user*** can be multi-part.

A RHS can also be preceded by a **$@** or a **$:** to control evaluation. A **$@** prefix causes the ruleset to return with the remainder of the RHS as the value. A **$:** prefix causes the rule to terminate immediately, but the ruleset to continue. This can avoid continued application of a rule. The prefix is stripped before continuing.

The **$@** and **$:** prefixes can precede a **$>** specification. For example, the form:

```
R$+     $:$>7$1
```

matches anything, passes that to ruleset seven, and continues; the **$:** is necessary to avoid an infinite loop.

Substitution occurs in the order described; that is, parameters from the LHS are substituted, hostnames are canonicalized, "subroutines" are called and, finally, **$#**, **$@**, and **$:** are processed.

Semantics of rewriting rule sets

There are five rewriting sets that have specific semantics. These are related as follows:

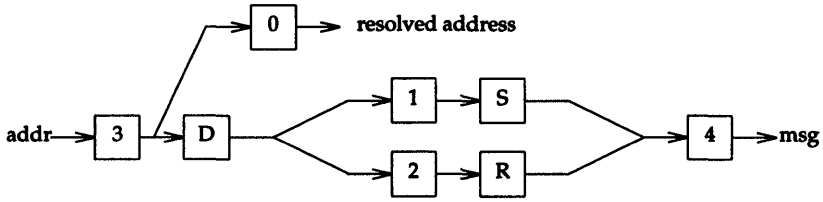


Figure 14-1 Rewriting set semantics

D - sender domain addition
S - mailer-specific sender rewriting
R - mailer-specific recipient rewriting

Ruleset three should turn the address into "canonical form." This form should have the basic syntax:

local-part@host-domain-spec

If no "@" sign is specified, then the host-domain-spec can be appended from the sender address (if the C flag is set in the mailer definition corresponding to the sending mailer). Ruleset three is applied by **sendmail** before doing anything with any address.

Ruleset zero is applied after ruleset three to addresses that are actually going to specify recipients. It must resolve to a {mailer, host, user} triple. The mailer must be defined in the mailer definitions from the configuration file. The host is defined into the **$h** macro for use in the argv expansion of the specified mailer.

Rulesets one and two are applied to all sender and recipient addresses, respectively. They are applied before any specification in the mailer definition. They must never resolve.

Ruleset four is applied to all addresses in the message. It is typically used to translate internal to external form.

Mailer flags

There are several flags that can be associated with each mailer, each identified by a letter of the alphabet. Many of them are assigned semantics internally. Any other flags can freely assign headers conditionally to messages destined for particular mailers.

The "error" mailer

The mailer with the special name "error" can generate a user error. The (optional) host field is a numeric exit status to be returned, and the user field is a message to be printed. For example:

```
$#error$:Host unknown in this domain
```

The entry on the RHS of a rule causes the specified error to be generated if the LHS matches. This mailer is only functional in ruleset zero.

Building a configuration file from scratch

Building a configuration file from scratch is an extremely difficult job. Fortunately, it is almost never necessary to do so; nearly every situation that may come up may be resolved by changing an existing table. In any case, it is critical that you understand what you are trying to do and come up with a philosophy for the configuration table. This section is intended to explain the real purpose of a configuration table and to give you some ideas as to what your philosophy might be.

Purpose of the configuration table

The configuration table has three major purposes. The first and simplest is to set up the environment for **sendmail**. This involves setting the options, defining a few critical macros, and so on. Because these are described in other sections, we will not go into more detail here.

The second purpose is to rewrite addresses in the message. This should typically be done in two phases. The first phase maps addresses in any format into a canonical form. This should be done in ruleset three. The second phase maps this canonical form into the syntax appropriate for the receiving mailer.

The **sendmail** program performs this second phase in the following three subphases: Rulesets one and two are applied to all sender and recipient addresses, respectively. After this, you can specify per-mailer rulesets for both sender and recipient addresses. This allows mailer-specific customization. Finally, ruleset four is applied to do any default conversion to external form.

The third purpose of the configuration table is to map addresses into the actual set of instructions necessary to get the message delivered. Ruleset zero must resolve to the internal form, which is in turn used as a pointer to a mailer descriptor. This describes the interface requirements of the mailer.

Relevant issues

The canonical form you use should almost certainly be as specified in the Arpanet standards documents RFC819 and RFC822. RFC822 describes the format of the mail message itself. The **sendmail** program follows this RFC

closely, to the extent that many of the standards described in this document cannot be changed without changing the code. In particular, the following characters have special interpretations:

< > ( ) " \

Any attempt to use these characters for other than their RFC822 purpose in addresses is probably doomed to disaster.

RFC819 describes the specifics of the domain-based addressing. This is touched on in RFC822 as well. Essentially, each host is given a name that is a right-to-left dot-qualified pseudo-path from a distinguished root. The elements of the path need not be physical hosts; the domain is logical rather than physical. For example, at Ocelot, one legal host might be "a.CC.Ocelot.EDU"; reading from right to left, "EDU" is a top level domain comprising educational institutions, "Ocelot" is a logical domain name, "CC" represents the Computer Center, (in this case a strictly logical entity), and "a" is a host in the Computer Center.

How to proceed

Once you have decided on a philosophy, it is worth examining the available configuration tables to decide whether any of them are close enough for you to use major parts of them as a basis for your configuration files. Even under the worst of conditions, there should be a large amount of material you can use.

The next step is to build ruleset three. This is the hardest part of the job. Beware of doing too much to the address in this ruleset, because anything you do reflects through to the message. In particular, stripping of local domains is best deferred, as this can leave you with addresses with no domain specifications at all. Because **sendmail** likes to append the sending domain to addresses with no domain, this can change the semantics of addresses. Also, try to avoid fully qualifying domains in this ruleset. Although technically legal, this can lead to unpleasantly and unnecessarily long addresses reflected into messages. The Ocelot configuration files define ruleset nine to qualify domain names and strip local domains. This is called from ruleset zero to get all addresses into a cleaner form.

Once you have ruleset three finished, the other rulesets should be relatively trivial. If you need hints, examine the supplied configuration tables.

Testing the rewriting rules: the -bt flag

When you build a configuration table, you can do a certain amount of testing using the "test mode" of **sendmail**. For example, you could invoke **sendmail** as:

sendmail -bt -Ctest.cf

This reads the configuration file *test.cf* and enter test mode. In this mode, you enter lines of the form:

rwset address

Here ***rwset*** is the rewriting set you want to use and ***address*** is an address to which to apply the set. Test mode shows you the steps it takes as it proceeds, finally showing you the address with which it ends up. You may use a comma-separated list of rwsets for sequential application of rules to an input; ruleset three is always applied first. For example:

1,21,4 monet:bollard

The first applies ruleset three to the input "monet:bollard." Ruleset one is then applied to the output of ruleset three, followed similarly by rulesets 21 and 4.

If you need more detail, you can also use the **-d21** flag to turn on more debugging. For example:

sendmail -bt -d21.99

This turns on a huge amount of information. A single-word address probably prints out several pages of information.

Building mailer descriptions

To add an outgoing mailer to your mail system, you must define the characteristics of the mailer.

Each mailer must have an internal name. This can be arbitrary, except that the names "local" and "prog" must be defined.

The pathname of the mailer must be given in the P field. If this mailer should be accessed via an IPC connection, use the string "[IPC]" instead.

The F field defines the mailer flags. You should specify an **"f"** or **"r"** flag to pass the name of the sender as a **-f** or **-r** flag, respectively. These flags are only passed if they were passed to **sendmail**, so that mailers that give errors under some circumstances can be placated. If the mailer is not picky, you can just specify "-f $g" in the argv template. If the mailer must be called as *root*, the **S** flag should be given. This does not reset the user ID before calling the mailer. If this mailer is local (that is, it performs final delivery rather than another network hop), the **l** flag should be given. Quote characters (backslashes and " marks) can be stripped from addresses if the **s** flag is specified. If this is not given, they are passed through. If the mailer is capable of sending to more than one user on the same host in a single transaction, the **m** flag should be stated. If this flag is on, then the argv template containing **$u** is repeated for each unique user on a given host. The **e** flag marks the mailer as being expensive, which causes **sendmail** to defer connection until a queue run.

An unusual case is the C flag. This flag applies to the mailer that the message is received from, rather than the mailer being sent to; if set, the domain specification of the sender (that is, the @host.domain part) is saved and is appended to any addresses in the message that do not already contain a domain specification. For example:

```
From: eric@ucbarpa
To: wnj@monet, mckee
```

A message of this form is modified to:

```
From: eric@ucbarpa
To: wnj@monet, mckee@ucbarpa
```

This happens if and only if the C flag is defined in the mailer corresponding to eric@ucbarpa.

The S and R fields in the mailer description are per-mailer rewriting sets to be applied to sender and recipient addresses, respectively. These are applied after the sending domain is appended and the general rewriting sets (numbers one and two) are applied, but before the output rewrite (ruleset four) is applied. A typical use is to append the current domain to addresses that do not already have a domain. For example:

```
From: eric
```

A header of this form might be changed to be:

```
From: eric@ucbarpa
```

Or to this form, depending on the domain it is being shipped into:

```
From: ucbvax!eric
```

These sets can also be used to do special-purpose output rewriting in cooperation with ruleset four.

The E field defines the string to use as an end-of-line indication. A string containing only newline is the default. The usual backslash escapes (\r, \n, \f, \b) may be used.

Finally, an argv template is given as the A field. It may have embedded spaces. If there is no argv with a **$u** macro in it, **sendmail** speaks SMTP to the mailer. If the pathname for this mailer is [IPC], the argv should be:

IPC $h [***port***]

Here ***port*** is the optional port number to connect to.

For example:

```
Mlocal, P=/usr/bin/mail, F=lsDFMmnPS S=10, R=20, A=lmail $u
Mether, P=[IPC], F=meC, S=11, R=21, A=IPC $h, M=100000
```

These specifications specify a mailer to do local delivery and a mailer for Ethernet delivery. The first is called local; it is located in the file */bin/mail*, takes a

picky **-r** flag, and does local delivery; quotes should be stripped from addresses, and multiple users can be delivered at once; ruleset 10 should be applied to sender addresses in the message, and ruleset 20 should be applied to recipient addresses. The argv to send to a message is the word "mail," the word "-d," and words containing the name of the receiving user. If a **-r** flag is inserted, it is between the words mail and -d. The second mailer is called ether; it should be connected via an IPC connection; it can handle multiple users at once; connections should be deferred; and any domain from the sender address should be appended to any receiver name without a domain. Sender addresses should be processed by ruleset 11 and recipient addresses by ruleset 21. There is a 100,000-byte limit on messages passed through this mailer.

Configuration options

The following options can be set using the **-o** flag on the command line or the **O** line in the configuration file. Many of them cannot be specified unless the invoking user is trusted.

A*file* Use the named *file* as the alias file. If no file is specified, use *aliases* in the current directory.

a*N* If set, wait up to *N* minutes for an @:@ entry to exist in the alias database before starting up. If it does not appear in *N* minutes, rebuild the database (if the **D** option is also set) or issue a warning.

B*c* Set the blank substitution character to *c*. Unquoted spaces in addresses are replaced by this character.

c If an outgoing mailer is marked as being expensive, do not connect immediately. This requires that queueing be compiled in, because it depends on a queue-run process to actually send the mail.

d*x* Deliver in mode *x*. Legal modes are:

- i Deliver interactively (synchronously).
- b Deliver in background (asynchronously).
- q Just queue the message (deliver during queue run).

D If set, rebuild the alias database if necessary and possible. If this option is not set, **sendmail** will never rebuild the alias database unless explicitly requested using **-bi**.

e*x* Dispose of errors using mode *x*. The values for *x* are:

- p Print error messages (default).
- q No messages, just give exit status.
- m Mail back errors.
- w Write back errors (mail if user not logged in).
- e Mail back errors and give zero exit status always.

F*n*	This is the temporary file mode, in octal. 644 and 600 are good choices.
f	Save UNIX-style "From" lines at the front of headers. Normally, they are assumed redundant and discarded.
g*n*	Set the default group ID for mailers to run into *n*.
H*file*	Specify the help file for SMTP.
I	Use the domain name server, **named**.
i	Ignore dots in incoming messages.
L*n*	Set the default log level to *n*.
M*x value*	Set the macro *x* to *value*. This is intended only for use from the command line.
m	Send to me, too, even if I am in an alias expansion.
N*netname*	The name of the home network (ARPA is the default). The argument of an SMTP HELO command is checked against *hostname.netname*, where *hostname* is requested from the kernel for the current connection. If they do not match, Received: lines are augmented by the name that is determined in this manner, so that messages can be traced accurately.
o	Assume that the headers may be in old format; that is, spaces delimit names. This actually turns on an adaptive algorithm: if any recipient address contains a comma, parentheses, or angle bracket, it is assumed that commas already exist. If this flag is not on, only commas delimit names. Headers are always output with commas between the names.
Q*dir*	Use the named *dir* as the queue directory.
q*factor*	Use *factor* as the multiplier in the map function to decide when just to queue up jobs rather than run them. This value is divided by the difference between the current load average and the load average limit (x flag) to determine the maximum message priority that is sent. Defaults to 10,000.
r*time*	Timeout reads after *time* interval.
S*file*	Log statistics in the named *file*.
s	Be extra safe when running things, that is, always instantiate the queue file, even if you are going to attempt immediate delivery. The **sendmail** program always instantiates the queue file before returning control to the client.
T*time*	Set the queue timeout to *time*. After this interval, messages that have not been successfully sent is returned to the sender.

tS,D — Set the local time zone name to *S* for standard time and *D* for daylight time. This is only used under version six.

u*n* — Set the default user ID for mailers to ***n***. Any mailer without the *S* flag in the mailer definition runs as this user.

v — Run in verbose mode.

w — Allow wildcard MX records.

x*LA* — When the system-load average exceeds *LA*, just queue messages (that is, do not try to send them).

X*LA* — When the system load average exceeds *LA*, refuse incoming SMTP connections.

y*fact* — The indicated ***fact***or is added to the priority (thus *lowering* the priority of the job) for each recipient, that is, this value penalizes jobs with large numbers of recipients.

Y — If set, deliver each job that is run from the queue in a separate process. Use this option if you are short of memory, because the default tends to consume considerable amounts of memory while the queue is being processed.

z*fact* — The indicated ***fact***or is multiplied by the message class (determined by the "Precedence:" field in the user header and the **P** lines in the configuration file) and subtracted from the priority. Thus, messages with a higher Priority is favored.

Z*fact* — The ***fact***or is added to the priority every time a job is processed. Thus, each time a job is processed, its priority is decreased by the indicated value. In most environments, this should be positive, because hosts that are down are all too often down for a long time.

Running sendmail

The following sections illustrate the flags and options used when executing **sendmail**, which is invoked from the */etc/tcp* shell script at startup. The *tcp* script is initialized by the **mkdev sendmail-init** and **mkdev cf** scripts, which are described earlier in this chapter. Refer to **tcp**(ADMN) for more information about */etc/tcp*.

Command line flags

Arguments must be presented with flags before addresses. The flags are:

-f ***addr*** — The sender's machine address is ***addr***. This flag is ignored unless the real user is listed as a "trusted user" or ***addr*** contains an exclamation point (because of certain restrictions in UUCP).

-r *addr* This flag is an obsolete form of **-f**.

-h *cnt* This flag sets the "hop count" to *cnt*. This represents the number of times this message has been processed by **sendmail** (to the extent that it is supported by the underlying networks). *cnt* is incremented during processing, and if it reaches MAXHOP (currently 30) **sendmail** throws away the message with an error.

-F*name* This flag sets the full name of this user to *name*.

-n Do not alias or forward.

-t Read the header for To:, Cc:, and Bcc: lines, and send to everyone in those lists. The Bcc: line is deleted before sending. Any addresses in the argument vector is deleted from the send list.

-b*x* Set operation mode to *x*. The operation modes are:

- m Deliver mail (default).
- a Run in ARPANET mode (see below).
- s Speak SMTP on input side.
- d Run as a daemon.
- t Run in test mode.
- v Just verify addresses, do not collect or deliver.
- i Initialize the alias database.
- p Print the mail queue.
- z Freeze the configuration file.

The special processing for the ARPANET includes reading the From: line from the header to find the sender, printing ARPANET-style messages (preceded by three-digit reply codes for compatibility with the FTP protocol), and ending lines of error messages with <CRLF>.

-q*time* Try to process the queued up mail. If the time is given, **sendmail** runs through the queue at the specified interval to deliver queued mail; otherwise, it only runs once.

-C*file* Use a different configuration file. The **sendmail** program runs as the invoking user (rather than *root*) when this flag is specified.

-d*level* Set debugging level.

-o*x value* Set option *x* to the specified *value*. These options are described in the next section.

There are a number of options that can be specified as primitive flags (provided for compatibility with **delivermail**). These are the **e**, **i**, **m**, and **v** options. Also, the **f** option can be specified as the **-s** flag.

Mailer flags

The following flags can be set in the mailer description.

f	The mailer wants a **-f** *from* **flag**, but only if this is a network forward operation (that is, the mailer gives an error if the executing user does not have special permissions).
r	This is the same as **f**, but sends a **-r** flag.
S	Do not reset the user ID before calling the mailer. This would be used in a secure environment where **sendmail** ran as *root*. This could be used to avoid forged addresses. This flag is suppressed if given from an unsafe environment (for example, a user's *mail.cf* file).
n	Do not insert a UNIX-system-style From: line on the front of the message.
l	This mailer is local (that is, final delivery is performed).
s	Strip quote characters off the address before calling the mailer.
m	This mailer can send to multiple users on the same host in one transaction. When a **$u** macro occurs in the *argv* part of the mailer definition, that field is repeated as necessary for all qualifying users.
F	This mailer wants a From: header line.
D	This mailer wants a Date: header line.
M	This mailer wants a Message-Id: header line.
x	This mailer wants a Full-Name: header line.
P	This mailer wants a Return-Path: line.
u	Uppercase should be preserved in user names for this mailer.
h	Uppercase should be preserved in host names for this mailer.
A	This is an Arpanet-compatible mailer, and all appropriate modes should be set.
U	This mailer wants UNIX-system-style From lines with the UUCP-style "remote from <host>" on the end.
e	This mailer is expensive to connect to, so try to avoid connecting normally. Any necessary connection occurs during a queue run.
X	This mailer wants to use the hidden dot algorithm as specified in RFC821. Basically, any line beginning with a dot has an extra dot prepended (to be stripped at the other end). This ensures that lines in the message containing a dot does not terminate the message prematurely.

L Limit the line lengths as specified in RFC821.

P Use the return-path in the SMTP "MAIL FROM:" command, rather than just the return address. Although this is required in RFC821, many hosts do not process return paths properly.

I This mailer is speaking SMTP to another **sendmail**. As such, it can use special protocol features. This option is not required (that is, if this option is omitted, the transmission still operates successfully, although perhaps not as efficiently as possible).

C If mail is received from a mailer with this flag set, any addresses in the header that do not have an at sign "@" after being rewritten by ruleset three has the @domain clause from the sender tacked on. This allows mail with headers of the form to be rewritten automatically:

```
From: usera@hosta
To: userb@hostb, userc
```

It becomes:

```
From: usera@hosta
To: userb@hostb, userc@hosta
```

E Escape lines beginning with "From" in the message with a ">" sign.

Arguments

Some important arguments of the **sendmail** program are described here.

Queue interval

The amount of time between forking a process to run through the queue is defined by the **-q** flag. If you run in mode **f** or **a**, this can be relatively large, because it is only relevant when a host that was down comes back up. If you run in **q** mode, it should be relatively short, because it defines the maximum amount of time that a message may sit in the queue.

Daemon mode

If you allow incoming mail over an IPC connection, you should have a daemon running. This should be set by your */etc/rc* file using the **-bd** flag. The **-bd** flag and the **-q** flag may be combined in one call:

/usr/lib/sendmail -bd -q30m

Forcing the queue

In some cases, you may find that the queue has gotten clogged for some reason. You can force a queue run using the **-q** flag (with no value). It is entertaining to use the **-v** flag (verbose) when this is done, to watch what happens:

/usr/lib/sendmail -q -v

Debugging

There is a fairly large number of debug flags built into **sendmail**. Each debug flag has a number and a level, where higher levels cause more information to be printed out. The convention is that levels greater than nine are not required. They print out so much information that you would not normally want to see them, except for debugging that particular piece of code. Debug flags are set using the **-d** option. The syntax is:

debug-flag:	**-d** debug-list
debug-list:	debug-option [, debug-option]
debug-option:	debug-range [. debug-level]
debug-range:	integer | integer - integer
debug-level:	integer

The spaces are for reading ease only. For example:

-d12	Set flag 12 to level 1.
-d12.3	Set flag 12 to level 3.
-d3-17	Set flags 3 through 17 to level 1.
-d3-17.4	Set flags 3 through 17 to level 4.

Trying a different configuration file

An alternative configuration file can be specified using the **-C** flag. The following example uses the configuration file *test.cf* instead of the default */usr/lib/sendmail.cf*.

```
/usr/lib/sendmail -Ctest.cf
```

If the **-C** flag has no value, it defaults to *sendmail.cf* in the current directory.

Changing the values of options

Options can be overridden using the **-o** flag. For example:

```
/usr/lib/sendmail -oT2m
```

This sets the **T** (timeout) option to two minutes for this run only.

Tuning

There are a number of configuration parameters you may want to change, depending on the requirements of your site. Most of these are set using an option in the configuration file. For example, the line "OT3d" sets option "T" to the value "3d" (three days).

Most of these options default appropriately for most sites. However, sites having very high mail loads may find they need to tune them as appropriate for their mail load. In particular, sites experiencing a large number of small messages, many of which are delivered to many recipients, may find that they need to adjust the parameters dealing with queue priorities.

Timeouts

All time intervals are set using a scaled syntax. For example, "10m" represents ten minutes, whereas "2h30m" represents two-and-a-half hours. The full set of scales is:

s	seconds
m	minutes
h	hours
d	days
w	weeks

Queue interval

The argument to the **-q** flag specifies how often a subdaemon runs the queue. This is typically set to between fifteen minutes and one hour.

Read timeouts

It is possible to time out when reading the standard input or when reading from a remote SMTP server. Technically, this is not acceptable within the published protocols. However, it might be appropriate to set it to something large (such as an hour) in certain environments. This reduces the chance of large numbers of idle daemons piling up on your system. This timeout is set using the **r** option in the configuration file.

Message timeouts

After sitting in the queue for a few days, a message times out. This means that if a message cannot be delivered for some reason, it is returned to the sender. This ensures that at least the sender is aware that the message was not sent. The timeout is typically set to three days. This timeout is set using the **T** option in the configuration file.

The time of submission is set in the queue, rather than the amount of time left until timeout. As a result, you can flush messages that have been hanging for a short period by running the queue with a short message timeout. For example:

```
/usr/lib/sendmail -oT1d -q
```

This runs the queue and flushes anything that is one day old.

Forking during queue runs

When the **Y** option is set, **sendmail** forks before each individual message while running the queue. This prevents **sendmail** from consuming large amounts of memory, and so it may be useful in memory-poor environments. However, if the **Y** option is not set, **sendmail** keeps track of hosts that are down during a queue run, which can improve performance dramatically.

Queue priorities

Every message is assigned a priority when it is first instantiated, consisting of the message size (in bytes) offset by the message class multiplied by the "work class factor" and the number of recipients multiplied by the "work recipient factor." The priority plus the creation time of the message (in seconds since January 1, 1970) are used to order the queue. Higher numbers for the priority mean that the message is processed later, when running the queue.

The message size is included so that large messages are penalized relative to small messages. The message class allows users to send high-priority messages by including a "Precedence:" field in their message; the value of this field is looked up in the **P** lines of the configuration file. Because the number of recipients affects the size of load a message presents to the system, this is also included in the priority.

The recipient and class factors can be set in the configuration file by using the **y** and **z** options, respectively. They default to 1000 (for the recipient factor) and 1800 (for the class factor). The initial priority is:

```
pri = size - (class * z) + (nrcpt * y)
```

(Remember, higher values for this parameter actually mean that the job is treated with lower priority.)

The priority of a job can also be adjusted each time it is processed (that is, each time an attempt is made to deliver it) using the "work time factor," set by the **Z** option. This is added to the priority, so it normally decreases the precedence of the job, on the grounds that jobs that have failed many times tend to fail again in the future.

Delivery mode

There are a number of delivery modes for **sendmail**, and they are set by the **d** configuration option. These modes specify how quickly mail is delivered. Legal modes are:

- i deliver interactively (synchronously)
- b deliver in background (asynchronously)
- q queue only (do not deliver)

There are tradeoffs. Mode "i" passes the maximum amount of information to the sender, but it is hardly ever necessary. Mode "q" puts the minimum load

on your machine, but means that delivery may be delayed for up to the queue interval. Mode "b" is probably a good compromise. However, this mode can cause large numbers of processes if you have a mailer that takes a long time to deliver a message.

File modes

There are several files involved with **sendmail** that can have a number of modes. The modes depend on the functionality you want and the level of security you require.

To suid or not to suid?

The **sendmail** program can safely be made setuid to *root*. At the point where it is about to **exec**(S) a mailer, it checks to see if the user ID is zero. If so, it resets the user ID and groupid to a default (set by the **u** and **g** options). (This can be overridden by setting the **S** flag to the mailer for mailers that are trusted and must be called as *root*.) However, this causes mail processing to be accounted to *root* rather than to the user sending the mail.

Temporary file modes

The mode of all temporary files that **sendmail** creates is determined by the **F** option. Reasonable values for this option are 0600 and 0644. If the more permissive mode is selected, it is not necessary to run **sendmail** as *root* at all (even when running the queue).

Should the alias database be writable?

The database that **sendmail** actually uses is represented by two files. These files are:

- *aliases.dir*
- *aliases.pag*

Both files are located in */usr/lib/mail*. The mode on these files should match the mode on */usr/lib/mail/aliases*. If *aliases* is writable and the DBM files (*aliases.dir* and *aliases.pag*) are not, users cannot reflect their desired changes through to the actual database. However, if *aliases* is read-only and the DBM files are writable, a slightly sophisticated user can arrange to steal mail anyway.

If your DBM files are not writable by the world or you do not have auto-rebuild enabled (with the **D** option), then you must be careful to reconstruct the alias database each time you change the text version via the command:

newaliases

If this step is ignored or forgotten, any intended changes are also ignored or forgotten.

Administering sendmail

System administration with **sendmail** consists of monitoring the system log file, managing the mail queue, maintaining the system alias file, and performing other related actions.

System log

The system log is entered in the file */usr/adm/syslog*. Each line in the system log consists of a timestamp, the name of the machine that generated it (for logging from several machines over the network), the word "sendmail," and a message.

A large amount of information can be logged. The log is arranged as a succession of levels. At the lowest level, only extremely strange situations are logged. At the highest level, even the most mundane and uninteresting events are recorded for posterity. As a convention, log levels under 10 are considered "useful"; log levels above 10 are usually for debugging purposes.

The mail queue

The mail queue should be processed transparently. However, you may find that manual intervention is sometimes necessary. For example, if a major host is down for a period of time the queue may become clogged. Although **sendmail** ought to recover gracefully when the host comes up, you may find performance unacceptably bad in the meantime.

The contents of the queue can be printed using the **mailq** command (or by specifying the **-bp** flag to **sendmail**):

```
mailq
```

This produces a listing of the queue identifiers, the size of the message, the date the message entered the queue, and the sender and recipients.

Format of sendmail queue files

All queue files have the form *x*f*AA99999*, where *AA99999* is the ID for this file and *x* is a type. The types are:

- d data file. The message body (excluding the header) is kept in this file.
- l lock file. If this file exists, the job is currently being processed, and a queue run does not process the file. For that reason, an extraneous *lf* file can cause a job to seem to disappear. It will not even time out.

n this file is created when an ID is being created. It is a separate file to ensure that no mail can ever be destroyed due to a race condition. It should exist for no more than a few milliseconds at any given time.

q queue control file. This file contains the information necessary to process the job.

t temporary file. This is an image of the *qf* file when it is being rebuilt. It should be renamed to a *qf* file very quickly.

x transcript file, existing during the life of a session, showing everything that happens during that session

The *qf* file is structured as a series of lines, each beginning with a code letter. The lines are as follows:

D name of the data file. There can only be one of these lines.

H header definition. There can be any number of these lines. The order is important: it represents the order in the final message. This uses the same syntax as header definitions in the configuration file.

R recipient address. This is normally completely aliased, but is actually re-aliased when the job is processed. There is one line for each recipient.

S sender address. There can only be one of these lines.

E error address. If any such lines exist, they represent the addresses that should receive error messages.

T job creation time. This computes how long a job remains in the queue undelivered, before being returned to the sender.

P current message priority. This orders the queue. Higher numbers mean lower priorities. The priority changes as the message sits in the queue. The initial priority depends on the message class and the size of the message.

M message. This line is printed by the **mailq** command, and is generally used to store status information. It can contain any text.

As an example, the following is a queue file sent to "mckee@calder" and "wnj:"

```
DdfA13557
Seric
T404261372
P132
Rmckee@calder
Rwnj
H?D?date: 23-Oct-82 15:49:32-PDT (Sat)
H?F?from: eric (Eric Allman)
H?x?full-name: Eric Allman
Hsubject: this is an example message
Hmessage-id: <8209232249.13557@UCBARPA.BERKELEY.ARPA>
Hreceived: by UCBARPA.BERKELEY.ARPA (3.227 [10/22/82])
    id A13557; 23-Oct-82 15:49:32-PDT (Sat)
HTo: mckee@calder, wnj
```

This shows the name of the data file, the person who sent the message, the submission time (in seconds since January 1, 1970), the message priority, the message class, the recipients, and the headers for the message.

Forcing the queue

The **sendmail** program should run the queue automatically at intervals. The algorithm is to read and sort the queue, then to attempt to process all jobs in order. When it attempts to run the job, **sendmail** first checks to see if the job is locked. If so, it ignores the job.

There is no attempt to ensure that only one queue processor exists at any time, because there is no guarantee that a job cannot take forever to process. Due to the locking algorithm, it is impossible for one job to freeze the queue. However, an uncooperative recipient host or a program recipient that never returns can accumulate many processes in your system. Unfortunately, there is no way to resolve this without violating the protocol.

In some cases, you may find that if a major host goes down for a couple of days, this can create a prohibitively large queue. This situation causes **sendmail** to spend an inordinate amount of time sorting the queue. This situation can be fixed by moving the queue to a temporary place and creating a new queue. The old queue can be run later, when the offending host returns to service.

To do this, it is acceptable to move the entire queue directory:

```
cd /usr/spool
mv mqueue omqueue; mkdir mqueue; chmod 777 mqueue
```

You should then kill the existing daemon (because it is still processing in the old queue directory) and create a new daemon.

To run the old mail queue, run the following command:

/usr/lib/sendmail -oQ/usr/spool/omqueue -q

The **-oQ** flag specifies an alternate queue directory, and the **-q** flag says just to run every job in the queue. Use the **-v** flag to view what is going on.

When the queue is finally emptied, you can remove the directory:

rmdir /usr/spool/omqueue

sendmail configuration file

Almost all configuration information for **sendmail** is read at runtime from the ASCII file */usr/lib/sendmail.cf*. This file has macro definitions encoded in it. These define such details as the value of macros used internally, header declarations, mailer definitions and address rewriting rules.

The **sendmail** configuration file is created as shown earlier in this chapter in the section "Configuring sendmail."

The header declarations tell **sendmail** the format of header lines that are processed specially. For example, any lines that are added or reformatted receive special processing. The mailer definitions give information such as the location and characteristics of each mailer. The address rewriting rules enable **sendmail** to be highly configurable and customizable, though this comes at the cost of some complexity.

The alias database

The alias database exists in two forms. One is a text form, maintained in the file */usr/lib/mail/aliases*. The aliases are of the form:

`name:` ***name1, name2, ...***

Only local names can be aliased. For example:

```
eric@mit-xx: eric@berkeley.EDU
```

This does not have the desired effect. Aliases can be continued by starting any continuation line with a space or a tab. Blank lines and lines beginning with a number sign "#" are comments.

The second form is processed by the **dbm**(S) library. This form is in the files */usr/lib/mail/aliases.dir* and */usr/lib/mail/aliases.pag*. This is the form that **sendmail** actually uses to resolve aliases. This technique improves performance.

Rebuilding the alias database

The DBM version of the database can be rebuilt explicitly by executing the command:

newaliases

This is equivalent to giving **sendmail** the **-bi** flag:

/usr/lib/sendmail -bi

If the "D" option is specified in the configuration, **sendmail** rebuilds the alias database automatically, if possible, when it is out of date. It does this under either or both of the following conditions:

- The DBM version of the database is mode 666.
- **sendmail** is running setuid to *root*.

Auto-rebuild can be dangerous on heavily loaded machines with large alias files; if it might take more than five minutes to rebuild the database, there is a chance that several processes start the rebuild process simultaneously.

Potential alias database problems

There are a number of problems that can occur with the alias database. They all result when a **sendmail** process accesses the DBM version while it is only partially built. This can happen under two circumstances: either one process accesses the database while another process is rebuilding it, or the process rebuilding the database dies (due to being killed or a system crash) before completing the rebuild.

The **sendmail** program includes two techniques to try to relieve these problems. First, it ignores interrupts while rebuilding the database; this avoids the problem of someone aborting the process and leaving a partially rebuilt database. Second, at the end of the rebuild it adds an alias of the form:

```
@: @
```

(Note that this is not normally legal.) Before **sendmail** accesses the database, it checks to ensure that this entry exists. The **sendmail** program waits for this entry to appear, at which point it forces a rebuild itself.

List owners

If an error occurs on sending to a certain address, say "*x*," **sendmail** looks for an alias of the form "owner-*x*" to receive the errors. This is typically useful for

a mailing list where the submitter of the list has no control over the maintenance of the list itself; in this case the list maintainer would be the owner of the list. For example:

```
unix-wizards: eric@ucbarpa, wnj@monet, nosuchuser,
    sam@matisse
owner-unix-wizards: eric@ucbarpa
```

This would cause "eric@ucbarpa" to get the error that occurs when someone sends to unix-wizards, due to the inclusion of "nosuchuser" on the list.

Per-user forwarding (.forward files)

As an alternative to the alias database, any user can put a file with the name *.forward* in his or her home directory. If this file exists, **sendmail** redirects mail for that user to the list of addresses listed in the *.forward* file. For example, suppose the home directory for user *mckee* has a *.forward* file with contents:

```
mckee@ernie
kirk@calder
```

Then any mail arriving for *mckee* is redirected to the specified accounts.

Special header lines

Several header lines have special interpretations defined by the configuration file. Others have interpretations built into **sendmail** that cannot be changed without changing the code. These built-ins are described here.

Return-receipt-to:

If this header is sent, a message is sent to any specified addresses when the final delivery is completed, that is, when successfully delivered to a mailer with the l flag (local delivery) set in the mailer descriptor.

Errors-to:

If errors occur anywhere during processing, this header causes error messages to go to the listed addresses rather than to the sender. This is intended for mailing lists.

Apparently-to:

If a message comes in with no recipients listed in the message (in a To:, Cc:, or Bcc: line), then **sendmail** adds an "Apparently-To:" header line for any recipients it is aware of. This is not put in as a standard recipient line to warn any recipients that the list is not complete.

Summary of support files

This is a summary of the support files that **sendmail** creates or generates.

/usr/lib/sendmail	binary of **sendmail**
/usr/bin/newaliases	link to */usr/lib/sendmail*; causes the alias database to be rebuilt. Running this program is completely equivalent to giving **sendmail** the **-bi** flag.
/usr/bin/mailq	prints a listing of the mail queue. This program is equivalent to using the **-bp** flag to **sendmail**.
/usr/lib/sendmail.cf	configuration file, in textual form
/usr/lib/sendmail.fc	configuration file represented as a memory image
/usr/lib/sendmail.hf	SMTP help file
/usr/lib/sendmail.st	statistics file; need not be present
/usr/lib/mail/aliases	textual version of the alias file
/usr/lib/mail/aliases.{pag,dir}	alias file in **dbm**(S) format
/usr/spool/mqueue	directory in which the mail queue and temporary files reside
*/usr/spool/mqueue/qf**	control (queue) files for messages
*/usr/spool/mqueue/df**	data files
*/usr/spool/mqueue/lf**	lock files
*/usr/spool/mqueue/tf**	temporary versions of the *qf* files, used during queue-file rebuild
*/usr/spool/mqueue/nf**	file used when creating a unique ID
*/usr/spool/mqueue/xf**	transcript of the current session

More sendmail information

The following manual pages provide additional information about **sendmail**:

Manual page	Description
aliases(SFF)	aliases file for **sendmail**
lmail(ADMN)	handle local mail delivery from **sendmail**
mailaddr(ADMN)	mail addressing description
mconnect(ADMN)	connect to SMTP mail server socket
rmail(ADMN)	handle remote mail received by UUCP
sendmail(ADMN)	**sendmail** command information
sendmail.cf(SFF)	**sendmail** configuration file

Chapter 15
Helpful hints

This chapter answers questions concerning:

- broadcast addresses
- problems with the WD8003 card
- remote backups
- **sendmail** program
- user equivalence across systems

Setting the broadcast address

Q: What should the host part of the broadcast address be?

A: The host part of the broadcast IP address should be all 0's with 4.1x or 4.2 BSD. In BSD 4.3, and versions derived from it such as SCO TCP/IP, the broadcast address has been corrected to use all 1's.

The INTERNET RFCs state that the host part of the IP address should be all 1's for broadcast purposes.

In releases prior to BSD 4.3, which were implemented before the standard was set, Berkeley implemented the broadcast address to be all 0's. Therefore, several Berkeley systems and TCP/IP implementations that are derived from BSD 4.1x and BSD 4.2 use all 0's for their broadcast address.

SCO supports both broadcast addresses; however, all hosts on a given network must have the same idea of what the broadcast address is. If you are running 4.3-derived TCP/IP, you still must use a broadcast address of 0's if

you want to broadcast on the same network as BSD 4.2-derived implementations. Consequently, in BSD 4.3 and implementations derived from BSD 4.3, there is a "broadcast" option to the **ifconfig** command that allows you to specify which broadcast address you are actually going to use.

ifconfig is run from the */etc/tcp* script. See **ifconfig**(ADMN) in the Network Commands (ADMN) section of the *SCO TCP/IP* manual for more details.

Problem with WD8003 card

Q: After installing SCO TCP/IP, I can **ping localhost** and send out packets, but I do not receive packets back from other machines. The WD8003 card is installed at interrupt vector 2, but it does not seem to be able to make the driver respond to its interrupts. If I **ping** another machine, I get 100% packet loss.

A: On some machines, the WD8003 card does not work at interrupt vector 2. This is because of a slightly non-standard bus specification on that machine.

You must reconfigure the device driver for the WD8003 card.

- Enter **netconfig**.
- Reconfigure the chain containing the card, and use interrupt vector 5. If there is a tape or a parallel port at that address, move it to vector 2.

Refer to the SCO TCP/IP *Release and Installation Notes* for instructions on reconfiguring device drivers.

You should test TCP/IP before adding any packages, such as NFS, on top of it. If you do add packages and need to reinstall the TCP/IP device driver, you will need to reinstall all the other packages as well.

Making remote backups

Q: How do I make a remote backup over an SCO TCP/IP network?

A: The following commands demonstrate how to archive files across a network. ***machinename*** should be replaced with the remote hostname.

When making a backup, you should keep the following points in mind:

- The machines should have host equivalency. This means that the SCO TCP/IP */etc/hosts.equiv* file must contain the appropriate hostnames that are involved in the backup. It may also be necessary to have a *$(HOME)/.rhosts* file depending on the UID of the user who is running the rcmd. See the SFF section of the SCO TCP/IP manual for more information on these files.
- If you are backing up whole filesystems, you should also consider that the **rcmd** command must be run as *root*, otherwise you do not have permission to read all the files.
- The **rcmd** command is executed from the current working directory. You may not want to include absolute pathnames in your backups. This means including a ./ or a . in the pathname specification, depending on what your current directory is.
- The examples shown cross mount points. That is, a filesystem that is mounted within the filesystem being backed up is also backed up.
- Although documented, **tar**(C) is not recommended for use as a backup mechanism because it does not save information about directories and special files. **tar** also cannot handle files whose names contain more than 100 characters. **cpio**(C) can handle filenames up to 256 characters in length.
- Neither **tar** nor **cpio** deals sensibly with sparse files (which cause a lot of data to be passed along the net).
- Backups on SCO TCP/IP for SCO UNIX System V/386 should only be made on SCO TCP/IP Release 1.1.1 or greater, as data corruption may occur. If you require a newer version, it is available as a free update.
- Backups should not be made on floppies, as they are limited in size and there is no easy way to load new disks as the backup continues. This may apply for small tape sizes as well.

Backing up files or filesystems

If you want to perform a backup where the command is issued locally, but the files and the device are remote, you may use one of the following commands:

rcmd *machinename* tar cvf /dev/rct0 /usr
rcmd *machinename* "find /usr -cpio -print | cpio -oc > /dev/rct0"

If the files are local, and the device is remote, you may use one of the following commands:

tar cvf - /usr | rcmd *machinename* dd of=/dev/rct0
find /usr -cpio -print | cpio -oc | rcmd *machinename* dd of=/dev/rct0

If the files are remote, and the device is local, you may use one of the following commands:

rcmd *machinename* "tar cvf - /usr" > /dev/rct0
rcmd *machinename* "find /usr -cpio -print | cpio -oc" > /dev/rct0

Restoring a backup

If you want to restore a backup where the command is issued locally, but the files and the device are remote, you may use one of the following commands. Note the changes to the **tar, cpio,** and **dd** commands.

rcmd *machinename* tar xvf /dev/rct0
rcmd *machinename* "cpio -idcv < /dev/rct0"

If the files are local, and the device is remote, you may use one of the following commands.

rcmd *machinename* dd if=/dev/rct0 | tar xvf -
rcmd *machinename* dd if=/dev/rct0 | cpio -idcv

If the files are remote, and the device is local, you may use one of the following commands:

dd if=/dev/rct0 | rcmd *machinename tar xvf* -
dd if=/dev/rct0 | rcmd *machinename* cpio -icdv

See the **tar, cpio,** and **rcmd**(C) manual pages for more information.

Differences in sendmail implementations

Q: What are the differences between the */usr/lib/sendmail* provided with SCO UNIX System V/386 Release 3.2 and the */usr/lib/sendmail* provided with SCO TCP/IP?

A: The */usr/lib/sendmail* program provided with SCO UNIX System V/386 Release 3.2 is a front end to the MMDF mail system. It is provided for those third party programs that normally use the Berkeley **sendmail** program as their mail agent.

The */usr/lib/sendmail* provided in SCO TCP/IP, and installed with **mkdev sendmail-init,** is the traditional Berkeley **sendmail** program. This **sendmail,** if installed with **mkdev sendmail-init,** completely replaces the MMDF mail system. As such, they are mutually exclusive. However, it must be noted that it is not absolutely necessary to use **sendmail** under SCO TCP/IP, as SCO TCP/IP and MMDF are compatible. Berkeley **sendmail** is provided for those more familiar with a traditional mailer.

Setting up user equivalence

Q: How do I set up user equivalence between two systems running SCO TCP/IP?

A: User equivalence is a state in which a particular user or group of users may access the accounts of another user or group of users, where this second group is usually on a different machine. This access is done without the use of any authentication, such as passwords.

For example, if user *alpha* on machine *m1* is equivalent to users *alpha* and *beta* on machine *m2*, then the following commands work without specifying any passwords from *alpha*'s account on *m1*. For example:

rlogin m2

User *alpha* logs into *m2*.

rlogin m2 -l beta

User *beta* logs into *m2*.

rcmd m2 who

User *alpha* executes the **who** command on machine *m2*.

rcmd m2 -l beta who

User *beta* executes the **who** command on machine *m2*.

Also, note the following:

rcp filename m2:filename2

This command requires user equivalence of user *alpha* on *m1* for user *alpha* on *m2*.

rcp filename beta@m2:filename2

This command requires user equivalence of user *alpha* on *m1* for user *beta* on *m2*.

There are two files that control this access. The first is *.rhosts* in the home directory of the affected user. The format of the *.rhosts* file is:

machine username

The username is optional.

The other file is */etc/hosts.equiv*. The format is identical to that of the *.rhosts* file, but usually only the ***machine*** portion is used.

For example, suppose user *alpha* on machine *m1* wants to allow user *alpha* on machine *m2* to access her account without the use of a password. User *alpha*, on *m1*, would create a file called *.rhosts* in her home directory with the line:

```
m2 alpha
```

In addition, if *alpha* wanted to allow user *delta* on *m2* and *gamma* on m3 to access her account without a password, the *.rhosts* file in *alpha*'s home directory would read:

```
m2 alpha
m2 delta
m3 gamma
```

If *alpha* additionally wanted all users on machine *m4* to be able to access her account without a password, the *.rhosts* file would read:

```
m2 alpha
m2 delta
m3 gamma
m4
```

Suppose that the system administrator of machine *m1* wanted to allow all users on machine *m5* to access their own accounts on machine *m1*. This would be accomplished by adding the following line to the */etc/hosts.equiv* file on *m1*:

```
m5
```

Thus user *beta* on machine *m5* could access user *beta* on *m1* without the need for a password.

Note that */etc/hosts.equiv* does not work for the user *root*. If you wish to access the *root* user on *m1* from *m2* without a password, you must set up a *.rhosts* file in the / directory on *m1*, with:

```
m2 root
```

Or, if you want a user other than *root* on *m2* to access *root* on machine *m1* without a password, use:

```
m2 user
```

Note that users on the machines with their own *.rhosts* file **must** have a password assigned. Also, if the system administrator has configured a */etc/hosts.equiv* file, the users on that system must have a password assigned to make use of the */etc/hosts.equiv* file. Finally, the *.rhosts* file in a particular user's home directory must be owned by that user.

Chapter 16
Bibliography

The following books provide additional useful information about TCP/IP that is beyond the scope of this *Administrator's Guide.*

- Comer, Douglas. *Internetworking With TCP/IP Volume I: Principles, Protocols, and Architecture,* 2nd edition. Prentice-Hall, Inc.: Englewood Cliffs, 1991

  A very accessible and more complete introduction to all aspects of TCP/IP and internetworking. This book includes information on: Internet Protocol (IP); User Datagram Protocol (UDP); Transmission Control Protocol (TCP); error and control Messages (ICMP); internet, subnet, and physical addressing; routing and gateways (EGP, RIP, OSPF, HELLO); client-server architecture; Bootstrap Protocol (BOOTP); network domains; Berkeley sockets; network applications; and internet management.

- Davidson, John. *An Introduction to TCP/IP.* Sprinter-Verlag: New York, 1988

  A brief overview of TCP/IP, organized by the seven layers of the OSI reference model. This book provides a good basic working knowledge of TCP/IP and how different parts of the protocol interact.

- Rose, Marshall. *The Simple Book.* Prentice-Hall: Englewood Cliffs, 1990.

  A thorough background reference on the Simple Network Management Protocol (SNMP). This book includes: definitions of the protocol, Structure of Management Information (SMI), and Management Information Base (MIB); examples of agent and management station configuration; and descriptions of common troubleshooting tasks you can accomplish using SNMP.

- Santifaller, Michael. *TCP/IP and NFS: Internetworking in a UNIX Environment.* Addison-Wesley (Deutschland) GmbH, 1991

Another overview of TCP/IP including protocol descriptions and common network administration tasks. Noteworthy in that it discusses NFS and its interaction with TCP/IP.

- Stevens, W. Richard. *UNIX Network Programming.* Prentice-Hall, Inc.: Englewood Cliffs 1991

  An excellent reference for network programming. This book includes information on: programming in sockets and TLI; how to program under the UNIX Operating System; descriptions of how daemon processes work (**fork** and **exec**); and how the TCP/IP protocols are implemented. Also included are some 15,000 lines of public domain source code, with examples of streams pipes, **rlogind**, **rcmd**, **lpd**, **tftp**, **rexec**, and **rmt**. This book covers issues not addressed in *"Internetworking with TCP/IP Volume 1."*

Index

Special characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Please help us to write computer manuals that meet your needs by completing this form. Please post the completed form to the Publications Manager nearest you: The Santa Cruz Operation, Ltd., Croxley Centre, Hatters Lane, Watford WD1 8YN, United Kingdom; The Santa Cruz Operation, Inc., 400 Encinal Street, P.O. Box 1900, Santa Cruz, California 95061, USA or SCO Canada, Inc., 130 Bloor Street West, 10th Floor, Toronto, Ontario, Canada M5S 1N5.

Volume title: _____

(Copy this from the title page of the manual, for example, SCO UNIX Operating System User's Guide)

Product: _____

(for example, SCO UNIX System V Release 3.2 Operating System Version 4.0)

How long have you used this product?

❑ Less than one month ❑ Less than six months ❑ Less than one year

❑ 1 to 2 years ❑ More than 2 years

How much have you read of this manual?

❑ Entire manual ❑ Specific chapters ❑ Used only for reference

	Agree				*Disagree*
The software was fully and accurately described	❑	❑	❑	❑	❑
The manual was well organized	❑	❑	❑	❑	❑
The writing was at an appropriate technical level (neither too complicated nor too simple)	❑	❑	❑	❑	❑
It was easy to find the information I was looking for	❑	❑	❑	❑	❑
Examples were clear and easy to follow	❑	❑	❑	❑	❑
Illustrations added to my understanding of the software	❑	❑	❑	❑	❑
I liked the page design of the manual	❑	❑	❑	❑	❑

If you have specific comments or if you have found specific inaccuracies, please report these on the back of this form or on a separate sheet of paper. In the case of inaccuracies, please list the relevant page number.

May we contact you further about how to improve SCO UNIX documentation? If so, please supply the following details:

Name _____ *Position* _____

Company _____

Address _____

City & Post/Zip Code _____

Country _____

Telephone _____ *Facsimile* _____