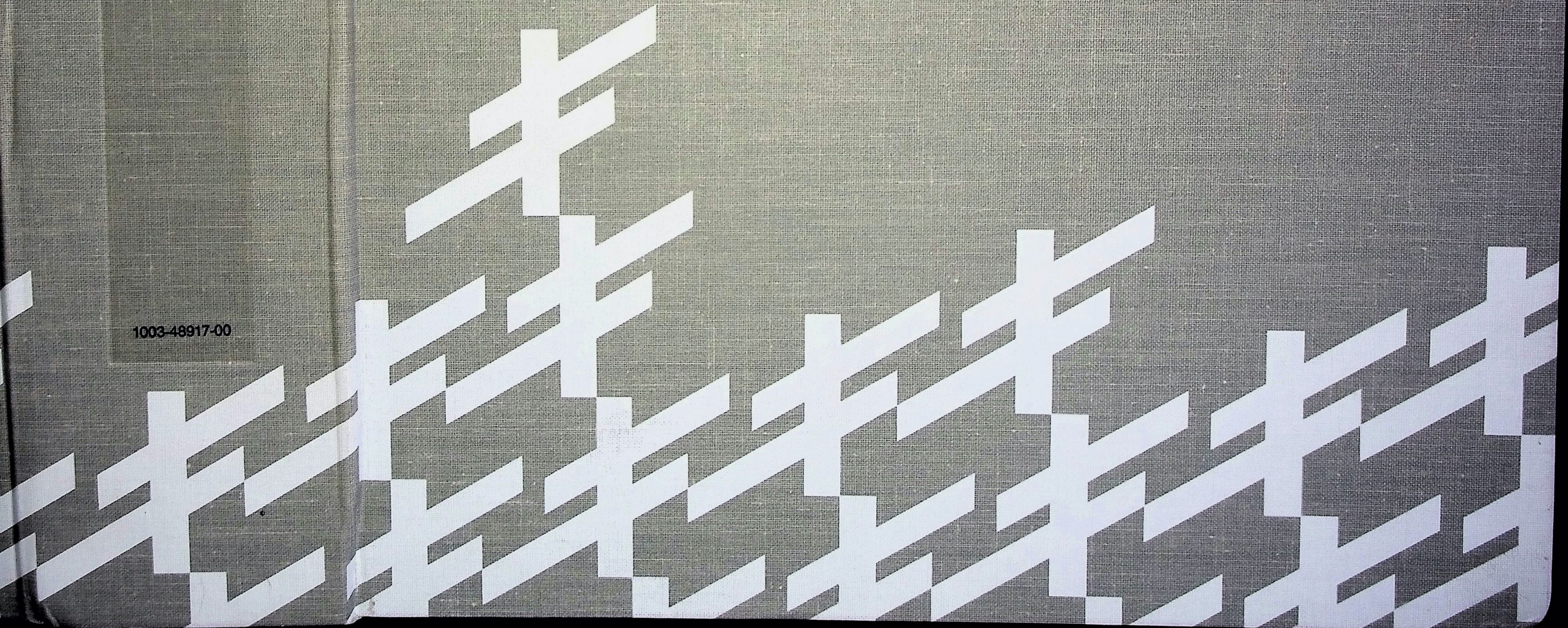
SEQUENT

ptx/TCP/IP

1003-48917-00

SEQUENT



THIS SHEET LIFTER
MUST BE PLACED ON
TOP OF CONTENTS

GRAPHIC PRODUCTS
Chicago, Illinois 60609

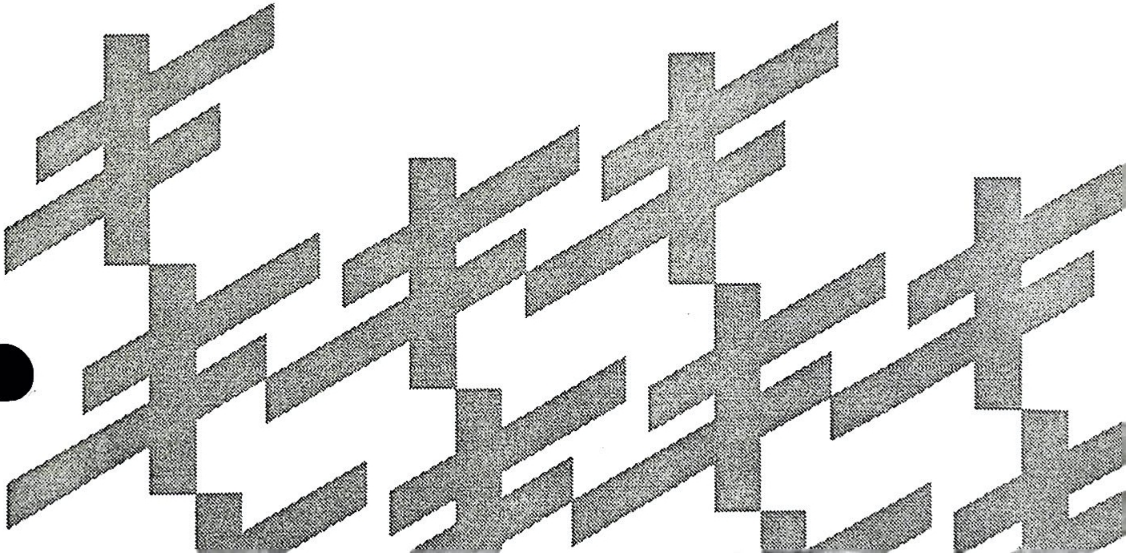
ptx/TCP/IP™

ptx/TCP/IP Administration Guide
1003-48918-00

ptx/TCP/IP Programming Manual
1003-48919-00

ptx/TCP/IP Reference Manual
1003-48920-00

Sequent Computer Systems, Inc.

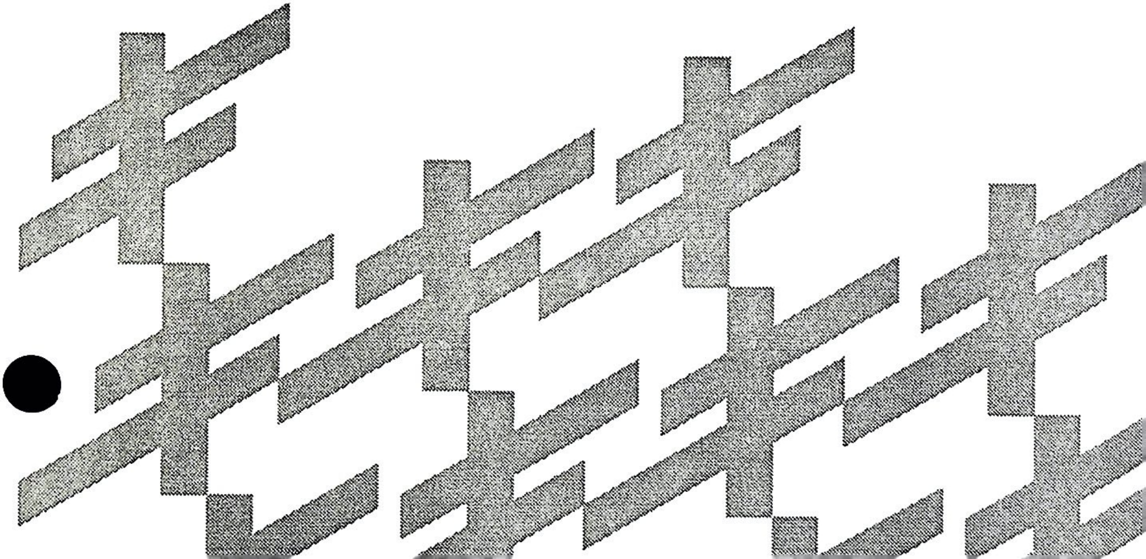


ptx/TCP/IP Administration Guide

ptx/TCP/IP Administration Guide

ptx/TCP/IP™
Administration Guide

1003-48918-00
Sequent Computer Systems, Inc.



Symmetry and Balance are registered trademarks of Sequent Computer Systems, Inc.

Ethernet is a trademark of Xerox Corporation.

UNIX is a registered trademark of AT&T.

Chapter 4, "Domain Name Server Configuration," is based on information presented in the *Name Server Operations Guide for BIND* by Kevin J. Dunlap.

Chapter 5, "Sendmail Configuration," is based on information presented in the articles *SENDMAIL – An Internet Mail Router* and *SENDMAIL Installation and Operation Guide* by Eric Allman.

Appendix B, "Name Server Operations Guide," contains the *Name Server Operations Guide for BIND* by Kevin J. Dunlap.

Appendix C, "Sendmail Overview," contains the *SENDMAIL – An Internetwork Mail Router* article by Eric Allman.

Appendix D, "Sendmail Installation and Operation Guide," contains the *SENDMAIL Installation and Operation Guide* by Eric Allman.

Copyright© 1990 by Sequent Computer Systems, Inc. All rights reserved. This document may not be copied or reproduced in any form without permission from Sequent Computer Systems, Inc. Information in this document is subject to change without notice.

Printed in the United States of America.

Contents

About This Guide

Assumptions About the Reader vii
Organization vii
Related Publications viii

Chapter 1. Administrative Concepts and Overview

1.1 Introduction 1-1
1.2 ptx/TCP/IP Processes and Files 1-5
1.3 Selecting an Internet Address 1-6
1.4 Configuration Prerequisites 1-9
1.5 Installation and Configuration Overview 1-9
1.6 Quickstart Configuration for a Single System 1-13

Chapter 2. Base Network Software Configuration

2.1 Introduction 2-1
2.2 Base Network Files 2-2
2.3 Starting ptx/TCP/IP 2-11
2.4 Performing Network Checks 2-12

Chapter 3. Network Administration

3.1 Introduction 3-1
3.2 Network Information Files 3-1
3.3 Network Permissions Files 3-6
3.4 Enabling rwhod 3-10
3.5 Network Utilities 3-10

Chapter 4. Domain Name Server Configuration

4.1 Introduction 4-1
4.2 Overview 4-1
4.3 Types of Servers 4-2
4.4 Name Server Files 4-3
4.5 Preparing Your System for the Name Server 4-6
4.6 Starting the Name Server 4-7
4.7 Sample Name Server Configuration 4-7

Chapter 5. Sendmail Configuration

5.1 Introduction 5-1
5.2 Sendmail Overview 5-1
5.3 Setting Up Sendmail 5-3

Appendix A. Configuration Worksheets

A.1 Introduction A-1
A.2 Base Network Software Worksheet A-1
A.3 Domain Name Server Worksheet A-3

Appendix B. Name Server Operations Guide

B.1 Introduction B-1
B.2 Building A System with a Name Server B-1
B.3 Types of Servers B-3
B.4 Setting up Your Own Domain B-4
B.5 Files B-5
B.6 Domain Management B-18

Appendix C. Sendmail Overview

C.1 Abstract C-1
C.2 Introduction C-1
C.3 Design Goals C-2
C.4 Overview C-4
C.5 Usage and Implementation C-7
C.6 Comparison With Other Mailers C-12
C.7 Evaluations and Future Plans C-14

Appendix D. Sendmail Installation and Operation Guide

D.1 Introduction D-1
D.2 Basic Installation D-2
D.3 Normal Operations D-6
D.4 Arguments D-13
D.5 Sendmail Tuning D-15
D.6 The Whole Scoop on The Configuration File D-20
D.7 Command Line Flags D-35
D.8 Configuration Options D-36
D.9 Mailer Flags D-40
D.10 Other Configuration D-41
D.11 Summary of Support Files D-47

Appendix E. Sendmail Configuration Files

E.1 Introduction E-1
E.2 TCP/IP Configuration File E-1
E.3 UUCP Configuration File E-12
E.4 TCP/IP and UUCP Configuration Files E-22

Appendix F. Removing the Installed ptx/TCP/IP Software

F.1 Introduction F-1
F.2 Preparing to Remove the Software F-1
F.3 Removing the Software F-1

Appendix G. Tuning the ptx/TCP/IP Kernel Parameters

Appendix H. Kernel Error Messages

H.1 Introduction H-1
H.2 WARNING Messages H-2
H.3 NOTE Messages H-6

Figures

1-1 Sample network 1-1
1-2 Internet address classes 1-6
1-3 ptx/TCP/IP configuration flowchart 1-11
1-3 ptx/TCP/IP configuration flowchart (cont) 1-12

2-1 Sample network composed of three networks 2-2

3-1 Network permissions flowchart 3-6

5-1 Sendmail structure 5-1

Tables

1-1 Network Classes 1-7

4-1 **named** Server Files 4-3

About This Guide

This guide tells the network or system administrator how to install and configure the ptx/TCP/IP and applications software. The ptx/TCP/IP software contains template files for the Berkeley Internet* Name Domain (BIND) Server and Sendmail Version 5.61.

Assumptions About the Reader

This guide assumes that the reader has basic UNIX system administration experience and is also familiar with the fundamentals of data communications and the TCP/IP protocol. For additional TCP/IP information, refer to the *ptx/TCP/IP and Applications Overview* listed in the "Related Publications" section.

Organization

This guide is organized as follows:

- Chapter 1, "Administrative Concepts and Overview," describes the concepts associated with network administration, lists configuration prerequisites, and provides an installation and configuration overview. This chapter also contains a step-by-step procedure for quickly bringing a system up on the network.
- Chapter 2, "Base Network Software Configuration," describes how to edit the base networking files.
- Chapter 3, "Network Administration," describes the network administration tasks including how to enable and disable the application services.
- Chapter 4, "Domain Name Server Configuration," describes how to obtain a subdomain name and how to configure the name server.
- Chapter 5, "Sendmail Configuration," describes how to install and configure the Sendmail facility.

---

* This guide uses the term *internet* (all lower case) to refer to any network that uses the TCP/IP protocol suite. The *Internet* (upper case I) is the wide area network that consists of a combination of networks such as the MILNET and the NSFNET. An internet may or may not be connected to the Internet.

- Appendix A, "Configuration Worksheets," contains worksheets for configuring the base software and domain name server.
- Appendix B, "Name Server Operations Guide," contains the *Name Server Operations Guide for BIND* by Kevin J. Dunlap.
- Appendix C, "Sendmail Overview," contains the *SENDMAIL – An Internetwork Mail Router* article by Eric Allman.
- Appendix D, "Sendmail Installation and Operation Guide," contains the *SENDMAIL Installation and Operation Guide* by Eric Allman.
- Appendix E, "Sendmail Configuration Files," shows the standard Sendmail configuration files that can be modified for your site.
- Appendix F, "Removing the Installed ptx/TCP/IP Software," describes the procedure for the orderly removal of the installed ptx/TCP/IP software.
- Appendix G, "Tuning the ptx/TCP/IP Kernel Parameters," lists the default ptx/TCP/IP kernel parameters and describes how to change the parameters through the ptx/ADMIN menu system.

Related Publications

The documents listed below provide additional information that may be helpful to the ptx/TCP/IP administrator:

- *ptx/TCP/IP and Applications Overview* Sequent Computer Systems, 1990, part number 1003-48916-00
- *DYNIX/ptx System Administration Guide*, Sequent Computer Systems, 1989, part number 1003-48609-00
- *DYNIX/ptx Operations Guide: ptx/ADMIN*, Sequent Computer Systems, 1989, part number 1003-48606-00

1. Concepts and Overview

1. Concepts and Overview

Chapter 1
Administrative Concepts and Overview

1.1 Introduction 1-1
  Protocols 1-3
  Client-Server Model 1-3
  Routing and Gateways 1-3
  Address Resolution 1-4
  Name Service 1-4
  Mail Routing 1-5
1.2 ptx/TCP/IP Processes and Files 1-5
1.3 Selecting an Internet Address 1-6
1.4 Configuration Prerequisites 1-9
1.5 Installation and Configuration Overview 1-9
1.6 Quickstart Configuration for a Single System 1-13

Figures

1-1 Sample network 1-1
1-2 Internet address classes 1-6
1-3 ptx/TCP/IP configuration flowchart 1-11
1-3 ptx/TCP/IP configuration flowchart (cont) 1-12

Tables

1-1 Network Classes 1-7

Chapter 1
Administrative Concepts and Overview

1.1 Introduction

This chapter introduces the user to the concepts of configuring and administering a system using the ptx/TCP/IP communication product. It also provides a quickstart procedure for configuring a SCED-based system on an Ethernet network.

Describing how to configure every network combination with the ptx/TCP/IP product would be an impractical task. Rather, a sample network has been created to illustrate essential configuration requirements. This network, shown in Figure 1-1, consists of three networks; administration, manufacturing, and engineering. Two gateway hosts; *mfg2* and *mfg3* connect the three networks. As will be shown later, the administration and manufacturing networks are actually subnetworks.

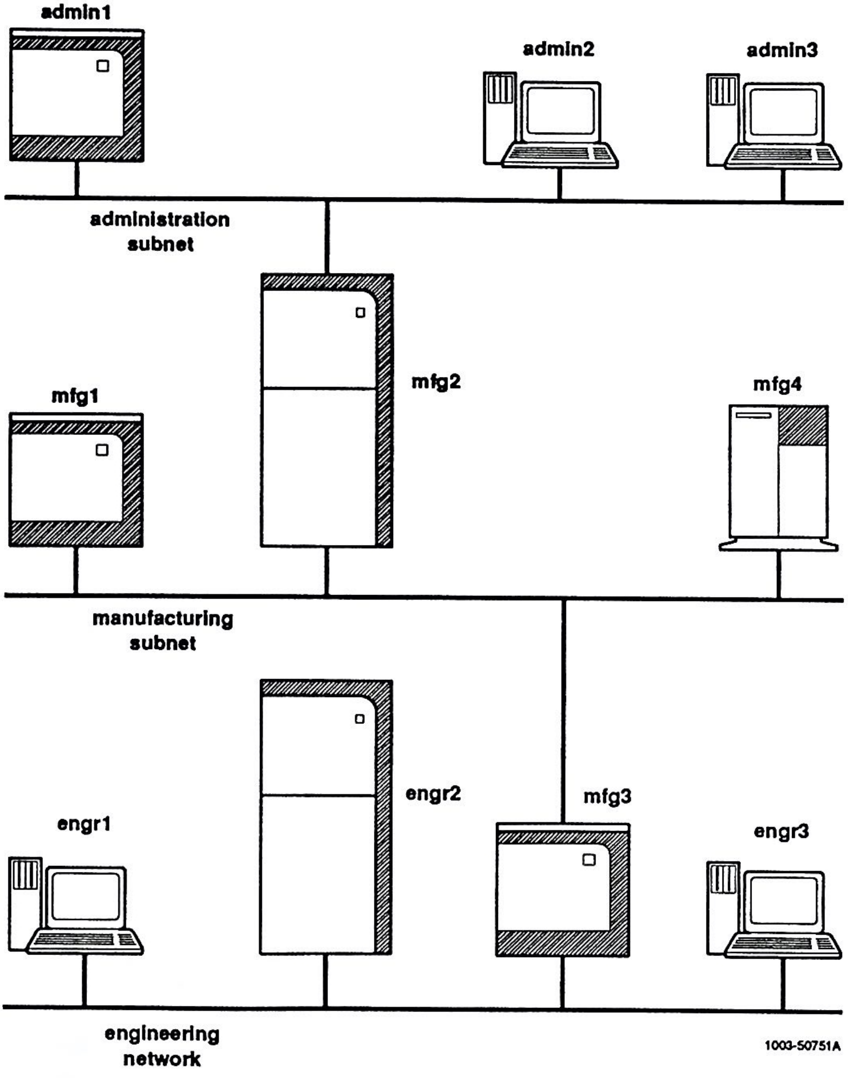

admin1
admin2
admin3
administration subnet
mfg2
mfg1
mfg4
manufacturing subnet
engr2
mfg3
engr1
engr3
engineering network
1003-50751A

Figure 1-1. Sample network. The sample network consists of three networks connected by two host gateways.

The following list describes concepts that are important to the configuration and administration of a TCP/IP network. Refer to the *ptx/TCP/IP and Applications Overview* for a detailed description of TCP/IP.

Protocols

TCP/IP consists of a number of protocols that specify message formats, establish connections, check for errors, control the flow of messages, and route messages to the proper destination. Each machine attached to a network (see Figure 1-1) may contain software that implements, among others, a combination of the following protocols:

- IP – internet protocol
- ICMP – internet control message protocol
- GGP – gateway-gateway protocol
- TCP – transmission control protocol
- UDP – user datagram protocol

The */etc/protocols* file specifies the protocols available on each machine.

Client-Server Model

The client-server model defines communication processes. A client process requests a service, generally over a network, from a server process. The server process in the form of an application program resides at a known IP address. Machines attached to a network may contain the application programs for a variety of server processes. For example, systems *admin2* and *admin3* in Figure 1-1 may have the file transfer protocol server (**ftpd**) running but not the **telnet** server (**telnetd**).

The */etc/inetd.conf* file tells the master server (**inetd**) what services to listen for on the network. The */etc/services* file maps a symbolic name to a numeric internet port number.

Routing and Gateways

Routing refers to the selection of a path for the data packets that travel between network machines. Machines on the same network maintain tables with the internet address of all other machines on the network. Sending a packet from one machine to another machine on the same network is a relatively straight forward task. For example, *engr1* in Figure 1-1 may have a packet for *engr2*. *engr1* compares the network portion of the packets internet address with that of *engr2*. Noting that the subnetwork portion of the addresses are identical, *engr1* routes the packet directly to *engr2*.

Packets for machines on other networks must be routed through a gateway. A gateway is a host that has a physical interface to at least two networks or subnetworks. *mfg3* and *mfg2* in Figure 1-1 act as gateways between the administration, manufacturing, and engineering networks. Routing tables in the gateway systems contain the network addresses of the subnetworks. The **route** utility allows the system administrator to modify the routing tables.

Address Resolution

TCP/IP software uses logical internet addresses, but the hardware connecting network components requires physical addresses. Thus, a packet transmitted from one machine to another requires that the internet address be translated to a physical address. The Address Resolution Protocol (ARP) consults an address resolution table to accomplish the translation. Typically, the ARP module automatically builds the address resolution table.

The **arp** utility allows the system administrator to display and modify the internet-to-Ethernet address resolution tables.

Name Service

A name service allows for a distributed database of host names and addresses. This is essential where a great number of machines attached to a network, such as the Internet, make it impractical for a single host to maintain all host names and addresses. With a name server, an entire network can be divided into subgroups known as *domains*. Hosts within a domain can then maintain the necessary host and address files. The name server is often referred to by its server daemon, **named**.

Configuring your system for the name server requires that you edit the following **named** files:

- */etc/named.boot*
- */etc/resolv.conf*
- */etc/named.local*
- */etc/named.hosts*
- */etc/named.hosts.rev*
- */etc/named.ca*

Chapter 4, "Domain Name Server Configuration," describes how to edit the **named** files and enable the name server on your system.

Mail Routing

Sendmail provides a general internetwork mail routing facility. It does not act as a mail-user interface but, rather handles aliasing, forwarding, and automatic routing of mail through gateways for a number of different mail interfaces.

Setting up a Sendmail facility from scratch is a complex task. Fortunately, the necessary files for a variety of configurations are included as part of the ptx/TCP/IP software. Chapter 5, "Sendmail Configuration," describes how to edit a Sendmail configuration file and enable the Sendmail facility.

1.2 ptx/TCP/IP Processes and Files

The following sequence describes how the ptx/TCP/IP processes, utilities, and files interact to establish communication on your system:

1. Install the ptx/TCP/IP software and compile the kernel.
2. The new kernel with the ptx/TCP/IP streams modules is installed when you reboot the system.
3. When you change the system from single-user (run level 1) to multiuser mode (run level 2), the kernel consults the */etc/rc2.d* directory and executes the files in order. When */etc/rc2.d/S50TCP* file is executed, it starts the **netd** utility. The *netd* utility reads the */etc/netconf* file to link the various ptx/TCP/IP streams devices and configure the network interfaces. The *etc/rc2.d/S50TCP* file is linked to the the*etc/init.d/tcp* file.
4. The */etc/rc2.d/S50netservers* file is executed to start the **inetd** process. The **inetd** process listens at the port associated with a supported network service such as **ftp** and **telnet**. It consults the */etc/inetd.conf* file to determine supported services and the */etc/services* file for the port number and protocol associated with a supported service. The *etc/rc2.d/S50netservers* file is linked to the the *etc/init.d/tcp* file.

   The */etc/rc2.d/S50netservers* file also starts the **syslog** facility by default and can be edited to start **rwhod, named**, and the Sendmail facility.

1.3 Selecting an Internet Address

Each host on a network has its own address that is unique to that network. This address, known as an *internet address*, identifies the packets sent across the network to the host. Whenever you add a new host to your network configuration, you will need to assign the host a new internet address.

An internet address has four 8-bit fields, each ranging in value from 0 to 255 (decimal). These four fields or 32-bits specify the two components of the internet address; the network address and a the host address. It is usually expressed as four decimal numbers separated by decimal points. For example, the 32-bit internet address:

```
01100001 00000000 00000000 00000100
```

would be expressed in the following decimal form:

```
97.0.0.4
```

In the example above, 97 specifies the network and 0.0.4 specifies the host on network 97. An internet address can also be represented as octal or hexadecimal. Decimal addresses have no prefixes and may not begin with zero. Octal addresses begin with a zero. Hexadecimal addresses are preceded by the 0x or 0X prefix. For example, 83.0.0.127 as a hexadecimal is 0x53.0.0.0x7F.

As shown in Figure 1-2, the network class A, B, or C determines how many fields comprise the network and host parts of the internet address.

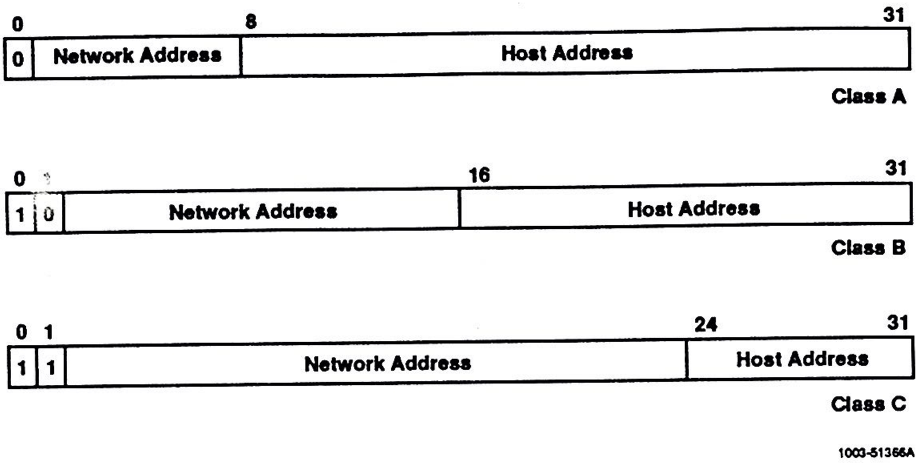

0
8
31
0
Network Address
Host Address
Class A
0
16
31
1
0
Network Address
Host Address
Class B
0
1
24
31
1
1
Network Address
Host Address
Class C
1003-51366A

Figure 1-2. Internet address classes. Internet addresses can be Class A, B, or C. The first part of the internet address is the network address and the second part is the host address.

Class A networks have an 8-bit network field with 0 as the most significant bit. Class B networks have a 16-bit network field with 10 as the two most significant bits. Class C networks have a 24-bit network field with 11 as the two most significant bits. The network field must be the same value for all hosts on the same network.

Table 1-1 shows the decimal and binary range of each network class.

Table 1-1
Network Classes

Decimal	Binary	Class
0-127	00000000 to 01111111	A
128-191	10000000 to 10111111	B
192-223	11000000 to 11011111	C

It is also possible to divide a network with a given network class, Class A, B or C, into distinct physical Ethernets. This is known as sub-netting with each part of the network known as a subnet. Sub-netting allows the connection of several Ethernets together as the same logical network, using only one network number. Most importantly, hosts on other networks see the group of subnets as one single network.

ptx/TCP/IP supports the division of a network into subnets, and acts as a gateway between two subnets and between a subnet and another network. ptx/TCP/IP can also forward packets from a host on a subnet to another gateway.

If you are attaching your system to an existing network, then you need to determine a host address only. If you a setting up a new network, you will need to determine which network class is best for your network. To do this you will need to consider how many hosts will be connected by the network and how many network segments you will need.

You should choose a Class A network if you are planning a network with many hosts on the same network. The 24 bits in the host part of a Class A network allows for a large number of different host numbers. Conversely, a Class C network gives you more scope for a large number of unique network numbers.

Class A is the most commonly used network class. Class C is least popular, although it is ideal for networks with point-to-point links.

If your network will be attached to the Internet, you must obtain a network number from the Network Information Center (NIC) at the following address:

Network Information Center
SRI International
333 Ravenswook Avenue
Menlo Park, California 94025
(800-235-3155)
(415-859-3695)

1.4 Configuration Prerequisites

Prior to installing and configuring the ptx/TCP/IP software on your system, we recommend that you meet the following prerequisites:

- **The correct version of the DYNIX/ptx operating system.** Check the *ptx/TCP/IP Release Notes* that were shipped with the ptx/TCP/IP documentation set to ensure that your system is running the correct version of the DYNIX/ptx operating system.
- **Network hardware and accompanying software properly installed.** You must have the required hardware and software installed on your system if you intend to use the ptx/TCP/IP product to communicate with another system. For example, systems connect to an Ethernet LAN through a SCSI/Ethernet/Diagnostics (SCED) board or a VMEbus Ethernet board. Both boards require a transceiver and transceiver cable. The ptx/LAN software must also be installed as described in the *ptx/LAN Administration Guide*.
- **Host name selected.** Ensure that your system has a name. Use the Set System Node Name menu to assign a name to your system.
- **Worksheet completed.** The base network worksheet shown in Appendix A should be completed before installing the ptx/TCP/IP software.

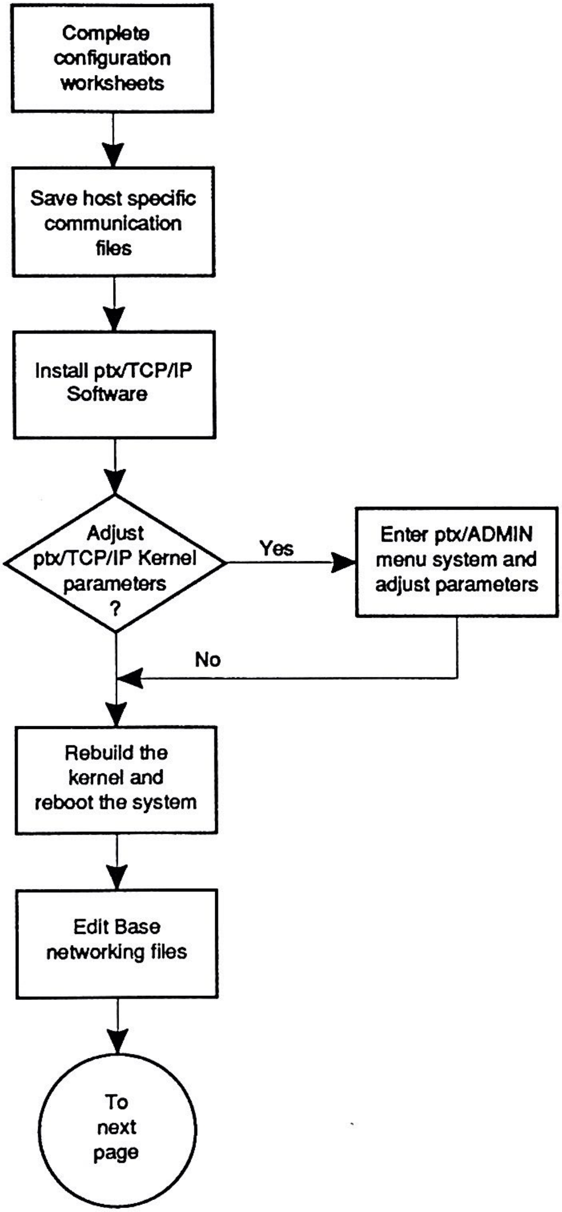
1.5 Installation and Configuration Overview

Installing and configuring ptx/TCP/IP on your system can be divided into the following tasks:

- **Complete the configuration worksheets.** Prior to installing the ptx/TCP/IP software, it is important to plan your system configuration The configuration worksheets in Appendix A, have been designed to aid with the configuration planning.
- **Save any host-specific communication files.** Your system may contain several TCP/IP files such as */etc/hosts*, */etc/protocols*, and */etc/services* that can be modified for use as part of the ptx/TCP/IP product. Save these files before installing the ptx/TCP/IP software.
- **Install the ptx/TCP/IP software.** The *ptx/TCP/IP Software Tape Instructions* packaged with the tape cartridge describe how to install the ptx/TCP/IP software.

- **If necessary, adjust any ptx/TCP/IP kernel parameters.** The default ptx/TCP/IP kernel parameters are suitable for most installations. Appendix G, "Tuning the ptx/TCP/IP Kernel Parameters," lists the default ptx/TCP/IP kernel parameters and describes how to change the parameters through the ptx/ADMIN menu system.
- **Rebuild the kernel.** Use the menu system to build a new kernel.
- **Reboot the system.** Reboot the system to the single-user mode.
- **Edit the base networking files.** The */etc/netconf*, */etc/init.d/tcp*, and */etc/hosts* files can be edited to allow the implementation of communication applications such as **rlogin** and **ftp**.
- **Edit the network administration files or copy the necessary files from another system on the network.** The network administration files listed in Chapter 4 specify the network services, permissions, and routing. These files can be created by editing the template files provided on the ptx/TCP/IP software tape or they can be copied from another system and modified for your system.
- **Configure the domain name server.** The domain name server is an optional facility. Using the name server requires the modification of several **named** template files.
- **Configure the Sendmail facility.** Sendmail is also an optional facility. A basic Sendmail configuration can be accomplished without modifying any files.
- **Change the system run level to the multiuser mode.** Using the Change System Run Level menu change the system to the multiuser mode.

The flowchart in Figure 1-2 shows the recommended sequence for performing the tasks listed above. In the following chapters, each task illustrated as part of the flowchart will be described completely.


Complete configuration worksheets
Save host specific communication files
Install ptx/TCP/IP Software
Adjust ptx/TCP/IP Kernel parameters ?
Yes
Enter ptx/ADMIN menu system and adjust parameters
No
Rebuild the kernel and reboot the system
Edit Base networking files
To next page

1003-50752-1A

Figure 1-3. ptx/TCP/IP configuration flowchart. The flowchart above and on the next page show the recommended sequence for configuring ptx/TCP/IP on your system.

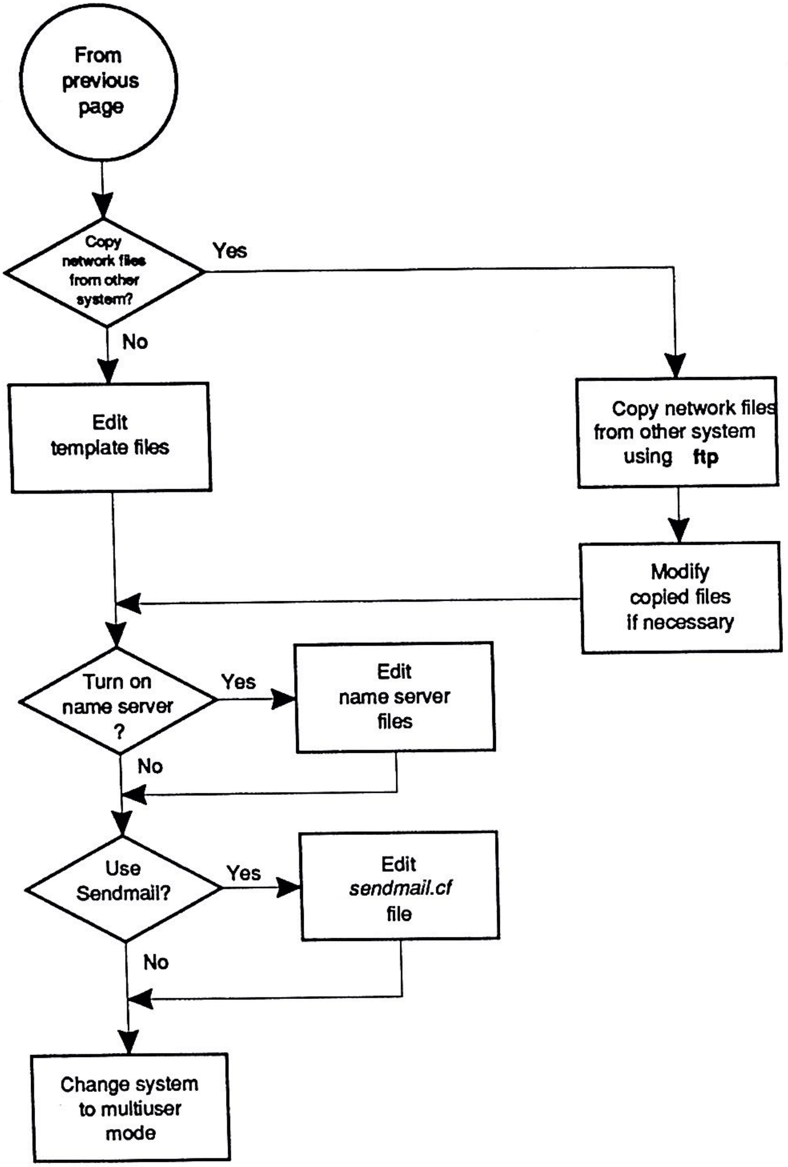

From previous page
Copy network files from other system?
Yes
No
Edit template files
Copy network files from other system using **ftp**
Modify copied files if necessary
Turn on name server ?
Yes
No
Edit name server files
Use Sendmail?
Yes
No
Edit *sendmail.cf* file
Change system to multiuser mode
1003-50752-2A

Figure 1-3. ptx/TCP/IP configuration flowchart (cont).

1.6 Quickstart Configuration for a Single System

This section provides a fast-track procedure for bringing a single SCED-based Ethernet system up on a network to communicate with another selected system. In other words, the following steps show you how to quickly set up your system to communicate with a single specified system on the same network. To illustrate this procedure, *mfg1* in Figure 1-1 will be configured to communicate with *mfg4*.

This procedure requires configuration of only two files: */etc/netconf* and */etc/hosts*.

1. **Ensure that your system is properly connected to the network.** *mfg1* contains a SCED board that connects to the Ethernet through a transceiver cable and a transceiver. The ptx/LAN software should installed and configured as described in the *ptx/LAN Administration Guide*
2. **Complete the Base Network Software Worksheet.** For our sample network, *mfg1* has been assigned an internet address of 101.2.0.1.
3. **Install the ptx/TCP/IP software.** Follow the tape instructions provided with the ptx/TCP/IP software tape.
4. **Rebuild the kernel and reboot the system.** Rebuild the kernel using the ptx/Admin menu system and reboot your system to the single-user run level.
5. **Edit the */etc/netconf* file.** The ptx/TCP/IP software contains a */etc/netconf* file. The following bold-face lines of the */etc/netconf* file have been modified for *mfg1*. Note that the comment characters (#) have been removed from the SCED-specific lines and that the IP address of *mfg1* has been added. Also note that the hexadecimal number 43 in the fifth field of the `ip` line specifies a 1's IP broadcast address.

```
#
# arp modules, one instance per ether device
#
arp0  m    arp
#
# uncomment arp1 if second ether connected to this
#  host
#
#arp1   m       arp
#
echo    dc      /dev/echo
#
# use these 2 entries for SCED based ethernets
```

```
#
eth0   dc      /dev/enet/se0
#eth1   dc      /dev/enet/se1
#
        .
        .
        .
#
# Sced ethernet 0,
#       1's IP broadcast, no subnet mask
#
ip    arp0   IP_NET={se0,101.2.0.1,,,43}  # <== editforlocaladdress ===
#
```

6. **Edit the *letc/hosts* file.** Add the name, IP address, and alias of your system and the name, IP address, and alias of a second system on the same network. For example, the */etc/hosts* file for *mfg1* has the following entries:

```
#
# Host Database
# This file should contain the addresses and aliases
# for local hosts that share this file.
#
# If the name server is running this file is not consulted
#
# Choose a distinguished local network address for localhost.
#
127.0.0.1          localhost
#
# Imaginary network
101.2.0.1          mfg1 host1 thishost
101.2.0.4          mfg4 host4
```

7. **Change the system run level to multiuser mode.** Using the ptx/ADMIN `Change System Run Level` menu, change your system to the multiuser mode. You should now be able to communicate with the selected system using applications such as **rlogin**, **rcp**, or **ftp**.

2. Base Network Configuration

2 Basic Network
Configuration

Chapter 2
Base Network Software Configuration

2.1 Introduction 2-1

2.2 Base Network Files 2-2
- /etc/netconf 2-2
  - One SCED Interface 2-5
  - Two SCED Interfaces 2-6
  - One SSM Interface 2-8
- /etc/init.d/tcp 2-9
- /etc/hosts 2-10

2.3 Starting ptx/TCP/IP 2-11

2.4 Performing Network Checks 2-12

Figures

2-1 Sample network composed of three networks 2-2

Chapter 2
Base Network Software Configuration

2.1 Introduction

As shown in Figure 1-2, after installing the ptx/TCP/IP software you should edit the following base networking files:

- */etc/netconf*
- */etc/init.d/tcp*
- */etc/hosts*

Editing these files enables communication applications such as **rlogin** and **ftp**. These applications then allow you to copy network information and permission files from another system.

Figure 2-1 (based on Figure 1-1) shows a sample network that will be used to illustrate how the base network files should be edited. The sample network consists of three networks; `admin_net, mfg_net, engr_net.` Each network has its own name and network address as does each system (host) on a network. Networks `admin_net` and `mfg_net` are actually subnetworks of network number 101. System administrators can select the numbers and names for all networks and hosts at a specific site. However, sites that connect to the Internet must contact the Network Information Center located at SRI International to receive network numbers.

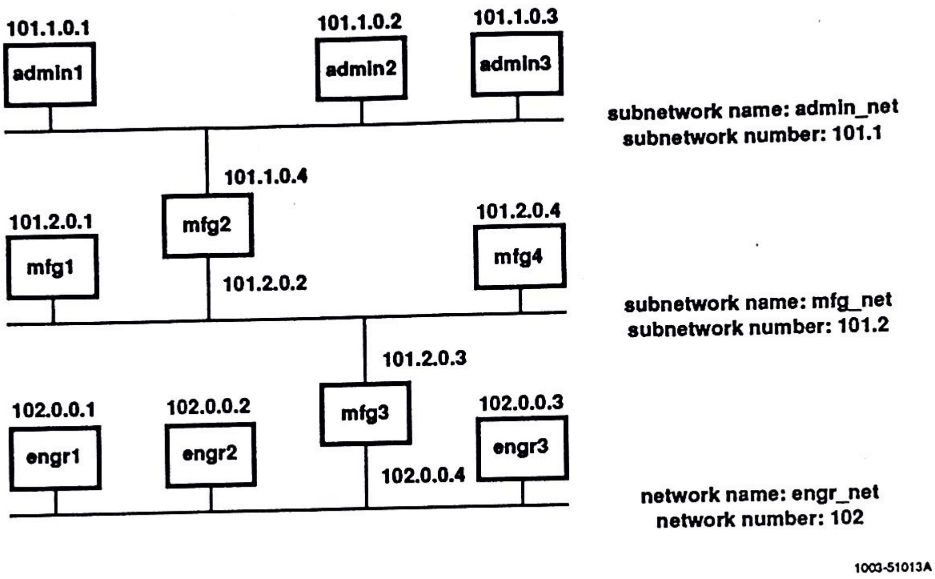

101.1.0.1
admin1
101.1.0.2
admin2
101.1.0.3
admin3
subnetwork name: admin_net
subnetwork number: 101.1
101.1.0.4
mfg2
101.2.0.1
mfg1
101.2.0.4
mfg4
101.2.0.2
subnetwork name: mfg_net
subnetwork number: 101.2
101.2.0.3
102.0.0.1
engr1
102.0.0.2
engr2
mfg3
102.0.0.3
engr3
102.0.0.4
network name: engr_net
network number: 102
1003-51013A

Figure 2-1. Sample network composed of three networks. The above network consists of a network connected to two subnetworks by gateway systems.

2.2 Base Network Files

The following sections describe the ptx/TCP/IP base networking files and describe how to these files should be edited for a particular system.

2.2.1 /etc/netconf

The */etc/netconf* file is used to build the internetwork streams system. It contains the name and internet address of your network interface. It also contains the names and relationships of all stream modules and drivers.

The ptx/TCP/IP software contains the following */etc/netconf* template file:

```
#
# etc/netconf
#
#  netd configuration file
#  used to build the internetwork streams system
#  and assign addresses to the network interfaces
#
#  If using SCED based ethernet use lines refering to
#  se0 and se1 devices
#
#  If using SSM based ethernet comment out all lines
#  refering to se0 and se1 devices and use eg0 and eg1
#  entries.
#
# name to driver/module mapping section
#
tcp dc /dev/tcp
udp dc /dev/udp
ip  dc /dev/ip
#
# arp modules, one instance per ether device
#
arp0   m  arp
#
# uncomment arp1 if second ether connected to this
#  host
#
#arp1  m  arp
#
echo   dc /dev/echo
#
# use these 2 entries for SCED based ethernets
#
eth0   dc /dev/enet/se0
#eth1  dc /dev/enet/se1
#
# use these 2 entries for ssm based ethernets
#
#eth0   dc    /dev/enet/eg0
#eth1  dc /dev/enet/eg1
#
loop   d  /dev/loop
bind    m       NULL

%%
#
# start of streams "wiring section"
# take the driver/module bindings
# defined above and connect them
# (do not remove preceeding %%)
#
# IP_NET fields are [1,2,3,4,5]
# 1 - device name (ascii)
# 2 - network address for this interface
# 3 - destination IP address for point to point links
# 4 - subnet mask
# 5 - configuration flags as follows
#
```

```
# valid flags for 5'th field of IP_NET
#
# IFF_UP            0x1 /* interface is up */
# IFF_BROADCAST  0x2 /* broadcast address is valid */
# IFF_DEBUG      0x4 /* turn on debugging (compat) */
# IFF_LOOPBACK      0x8 /* is a loopback net */
# IFF_POINTTOPOINT  0x10   /* interface is point to point (like slip) */
# IFF_NOTRAILERS 0x20   /* avoid trailers */
# IFF_RUNNING    0x40   /* resources allocated */
# IFF_ZEROBROADCAST 0x80   /* use 0's in broadcast address */
tcpip
udpip
bind    eth0    ECHO_TYPE
echo   bind
arp0   eth0   ARP_TYPE
#
# uncomment this entry for second ethernet
#
#arp1  eth1   ARP_TYPE
#
# Sced ethernet 0,
#     1's IP broadcast, no subnet mask
#
ip arp0   IP_NET={se0,97.0.0.1,,,43} # <== editforlocaladdress ===
#
# uncomment for second SCED ethernet
#
#iparp1   IP_NET={se1,98.0.0.1,,,43} # <== editforlocaladdress ===
#
# SSM ethernet 0
#     1's IP broadcast, no subnet mask
#
#iparp0   IP_NET={eg0,97.0.0.1,,,43}
#
# uncomment for second SSM ethernet
#
#iparp1   IP_NET={eg1,98.0.0.1,,,43}
#
# loopback device.  Do not remove or change
# address
#
ip loop   IP_NET={loop,127.0.0.1,,,ed}
```

The `IP_NET` parameter contains five fields that are defined in commented sections of the template file. Change field 2 of the `IP_NET` parameter to the IP address of your host. For example, the `admin_net` subnetwork shown in Figure 2-1 has a network address of 101.1. As shown below, each host on the network has a host address that is unique to subnetwork 101.1.

```
Machine       Address
```

```
-------       -------
admin1        101.1.0.1
admin2        101.1.0.2
admin3        101.1.0.3
```

The */etc/netconf* file can be edited to support the maximum number of SCEDs or SSMs allowed in a system.

The following three sections show a base networking configuration worksheet and the resulting */etc/netconf* file for each of the following system hardware combinations:

- One SCED interface
- Two SCED interfaces
- One System Service Module (SSM) with a VMEbus Ethernet interface

One SCED Interface

A system with a single SCED board connects to an Ethernet through a transceiver cable and transceiver. A completed base network configuration worksheet (see Appendix A) for a single-SCED system would be similar to that shown below for *engr2* in Figure 2-1. The default gateway address for *mfg3* is entered in the *etc/init.d/tcp* file.

System Name____engr2______

Prim Addr__102.0.0.2_____ Interface_se0__ Subnet Mask___________ IP Brdcst_1's_

IP Addr #1________________ Interface_____ Subnet Mask___________ IP Brdcst_____

IP Addr #2________________ Interface_____ Subnet Mask___________ IP Brdcst_____

IP Addr #3________________ Interface_____ Subnet Mask___________ IP Brdcst_____

Default Gateway___mfg3______

IP Forwarding [on/off]____on_______

Name server [on/off]_____off____

The following lines of the */etc/netconf* file for *engr2* would be edited as shown below:

```
#
# arp modules, one instance per ether device
#
arp0    m  arp
#
        .
        .
        .
#
# use these 2 entries for SCED based ethernets
#
eth0    dc  /dev/enet/se0
#eth1    dc  /dev/enet/se1
#
        .
        .
        .
#
# Sced ethernet 0,
#    1's IP broadcast, no subnet mask
#
ip   arp0    IP_NET={se0,102.0.0.2,,,43} # <== editforlocaladdress ===
#
```

Two SCED Interfaces

A system with two SCED boards can be used as a gateway between two networks. A completed base network configuration worksheet (see Appendix A) for a two-SCED system would be similar to that shown below for *mfg3* in Figure 2-1. Note that an IP broadcast address setting of 1's (ones) is standard. A 0's (zeros) IP broadcast address is necessary for those sites requiring backword compatibility.

System Name ____mfg3______

Prim Addr__102.0.0.4____Interface_se0_ Subnet Mask______________ IP Brdcst_1's_

IP Addr #1__101.2.0.3__ Interface_se1_ Subnet Mask_255.255.0.0__ IP Brdcst_0's_

IP Addr #2________________ Interface_____ Subnet Mask___________ IP Brdcst_____

IP Addr #3________________ Interface_____ Subnet Mask___________ IP Brdcst_____

Default Gateway__mfg2______

IP Forwarding [on/off]____on_______

Name server [on/off]_____off____

The following lines of the */etc/netconf* file for *mfg3* would be edited as shown below:

```
#
# arp modules, one instance per ether device
#
arp0    m   arp
#
# uncomment arp1 if second ether connected to this
#  host
#
arp1    m   arp
#
# use these 2 entries for SCED based ethernets
#
eth0    dc  /dev/enet/se0
eth1    dc  /dev/enet/se1
#
        .
        .
        .
arp0    eth0    ARP_TYPE
arp1    eth1    ARP_TYPE
#
# uncomment this entry for second ethernet
#
arp1    eth1    ARP_TYPE
#
# Sced ethernet 0,
#    1's IP broadcast, no subnet mask
#
ip   arp0    IP_NET={se0,102.0.0.4,,,43}  # <== editforlocaladdress ===
#
# uncomment for second SCED ethernet
#
ip   arp1    IP_NET={se1,101.2.0.3,,255.255.0.0,c3}    # <== editforlocaladdress ===
#
```

One SSM Interface

A system with an SSM board connects to an Ethernet through a VMEbus Ethernet board, a transceiver cable, and a transceiver. A completed base network configuration worksheet (see Appendix A) for a single SSM system would be similar to that shown below for *mfg4* in Figure 2-1.

System Name _____mfg4____

Prim Addr__101.2.0.4___ Interface_eg0_ Subnet Mask_255.255.0.0__ IP Brdcst_1's_

IP Addr #1______________ Interface_____ Subnet Mask_____________ IP Brdcst_____

IP Addr #2______________ Interface_____ Subnet Mask_____________ IP Brdcst_____

IP Addr #3______________ Interface_____ Subnet Mask_____________ IP Brdcst_____

Default Gateway____mfg2_____

IP Forwarding [on/off]____off______

Name server [on/off]_____off____

The following lines of the */etc/netconf* file for *mfg3* would be edited as shown below. Note that any lines referring to a SCED (se) Ethernet device have been commented out.

```
#
# arp modules, one instance per ether device
#
arp0    m   arp
#
     .
     .
     .
#
# use these 2 entries for SCED based ethernets
#
#eth0   dc  /dev/enet/se0
#eth1   dc  /dev/enet/se1
#
# use these 2 entries for ssm based ethernets
#
```

```
eth0  dc    /dev/enet/eg0
#eth1   dc  /dev/enet/eg1
#
      .
      .
      .
#
# Sced ethernet 0,
#    1's IP broadcast, no subnet mask
#
#ip arp0     IP_NET={se0,97.0.0.1,,,43}    # <== editforlocaladdress ===
#
# uncomment for second SCED ethernet
#
#ip arp1     IP_NET={se1,98.0.0.1,,,43}    # <== editforlocaladdress ===
#
# SSM ethernet 0
#    1's IP broadcast, no subnet mask
#
ip   arp0    IP_NET={eg0,101.2.0.4,,255.255.0.0,c3}
#
```

2.2.2 /etc/init.d/tcp

The */etc/init.d/tcp* file is linked to the level 2 *init* startup routine */etc/rc2.d/S50TCP*. This file specifies gateway routing information.

The ptx/TCP/IP software contains the following */etc/init.d/tcp* template file:

```
#
# Start the tcp/ip network system
#
# netd - reads /etc/netconf to link the streams
#      devices and configure network
#      interfaces
/etc/netd
#
# Start servers specified in /etc/inetd.conf
#
/etc/inetd
#
# change this entry for default gateway.
#
# add any additional routes here.
#
#route add net default 97.0.0.99 1
```

Edit the following line to specify the internet address of your default gateway host. Be sure to remove the comment character (#).

```
route add net default 97.0.0.99 1
```

admin1 shown in Figure 2-1 could have the following */etc/init.d/tcp* file:

```
#
# Start the tcp/ip network system
#
# netd - reads /etc/netconf to link the streams
#      devices and configure network
#      interfaces
/etc/netd
#
# Start servers specified in /etc/inetd.conf
#
/etc/inetd
#
# change this entry for default gateway.
#
# add any additional routes here.
# (Post V0.1 have route read /etc/gateways file)
#
route add net default 101.1.0.4 1
```

2.2.3 /etc/hosts

The */etc/hosts* file contains a list of the hosts on your network. and, if desired, gateways and hosts on other networks. Each file entry includes the host's internet address, its official name, and its alias or nickname. The following entry from the template */etc/hosts* file shows the correct line format for each host:

```
97.0.0.2      host2.my.domain host2
```

Adding a host to the network requires an entry in the above format. Removing a host's */etc/hosts* file entry removes the host's name from the network. Access to a host with no entry in the*/etc/hosts* file, requires the host's IP address. Upon starting, each network host reads the */etc/hosts* file to determine its own IP address.

The ptx/TCP/IP software contains the following */etc/hosts* template file:

```
#
# Host Database
# This file should contain the addresses and aliases
# for local hosts that share this file.
#
```

```
# If the name server is running this file is not consulted
#
# Choose a distinguished local network address for localhost.
#
127.0.0.1     localhost localhost.my.domain
#
# Imaginary network
97.0.0.2      host2.my.domain host2
97.0.0.3      host3.my.domain host3
```

The */etc/hosts* file for *admin1* in Figure 2-1 could have the following entries:

```
#
# Host Database
# This file should contain the addresses and aliases
# for local hosts that share this file.
#
# If the name server is running this file is not consulted
#
# Choose a distinguished local network address for localhost.
#
127.0.0.1          localhost
#
# administration subnetwork
#
101.1.0.2      admin2 host1
101.1.0.3      admin3 host3
#
# manufacturing subnetwork gateway machine
#
101.1.0.4          mfg2
```

2.3 Starting ptx/TCP/IP

To start ptx/TCP/IP, change the system run level to multiuser mode. Use the ptx/ADMIN `Change System Run Level` menu to change to the multiuser mode.

2.4 Performing Network Checks

After changing to the multiuser mode, check the communications facilities of your machine. The following sequence describes the recommended procedure for testing your system:

1. **ping** to yourself.
2. **ping** another host on the network.
3. **rlogin** to yourself.
4. **rlogin** to another host.
5. **ftp** a file to or from another host.

3. Network Administration

3. Network Administration

Chapter 3
Network Administration

3.1 Introduction 3-1

3.2 Network Information Files 3-1
- /etc/networks 3-1
- /etc/protocols 3-2
- /etc/services 3-3
- /etc/inetd.conf 3-4
- /usr/hosts/* 3-6

3.3 Network Permissions Files 3-6
- /etc/hosts.equiv 3-8
- .rhosts 3-9
- /etc/ftpusers 3-9

3.4 Enabling rwhod 3-10

3.5 Network Utilities 3-10
- arp 3-10
- netstat 3-11
- ping 3-13
- route 3-14

Figures

3-1 Network permissions flowchart 3-6

Chapter 3
Network Administration

3.1 Introduction

This chapter describes the network information and network permissions files. It also describes how to setup a Sequent system as a gateway to another network and how to use the various network utilities.

The ptx/TCP/IP software contains templates for the various network information and permissions files. You may have copied these files from another system after configuring the base network software.

3.2 Network Information Files

The network information files provide listings of systems (hosts) on the network, networks and subnetworks, protocols supported, and services supported. The following is a list of network information files:

- */etc/networks*
- */etc/protocols*
- */etc/services*
- */etc/inetd.conf*
- */usr/hosts/**

3.2.1 /etc/networks

The */etc/networks* file lists the name of each network that can be reached by a host on your network. This file, typically the same for all systems, associates a symbolic name with the Internet network number of each subnetwork. Applications such as **netstat** and **route** use the */etc/networks* to translate between network names and their associated network internet address.

The */etc/networks* file for each system shown in Figure 2-1, would have the following entries:

```
admin_net    101.1
mfg_net      101.2
engr_net     102
```

Unless you are adding or deleting a network, you do not need to change the */etc/networks* file that you copied from another system.

3.2.2 /etc/protocols

The */etc/protocols* lists the protocols used in the Internet. Each protocol requires a file entry with the following information:

official protocol name   protocol number   aliases

The protocol names must not contain spaces, tabs, newlines, or number (#) characters. Spaces or tabs must be used to separate fields within each entry. Number characters (#) are used to comment a line.

You do not need to modify the */etc/protocols* file unless you want to add a non-standard internet service to your local area network.

The */etc/protocols* template file contains the following entries:

```
#
# /etc/protocols
#
# Internet (IP) protocols
#
ip       0       IP       # internet protocol, pseudo protocol number
icmp     1       ICMP     # internet control message protocol
ggp      3       GGP      # gateway-gateway protocol
tcp      6       TCP      # transmission control protocol
egp      8       EGP      # exterior gateway protocol
pup      12      PUP      # PARC universal packet protocol
udp      17      UDP      # user datagram protocol
hmp      20      HMP      # host monitoring protocol
xns-idp  22      XNS-IDP  # Xerox NS IDP
rdp      27      RDP      #"reliable datagram" protocol
```

3.2.3 /etc/services

The */etc/services* file lists services available on the Internet. Each service requires a file entry with the following information:

official service name   port number/protocol name   aliases

The service names must not contain spaces, tabs, newlines or number (#) characters. Any other printable character is acceptable. Spaces or tabs must be used to separate fields within each entry. The port number and the protocol name are parts of the same field, delimited by a (/) character. For example, `514/tcp`. Comments must be preceded with a number (#) character.

The */etc/services* template file contains the following entries:

```
#
# etc/services
#
#
# Network services, Internet style
#
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udpsink       null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/udp          resource        # resource location
nameserver      42/tcp          name            # IEN 116
whois           43/tcp          nicname
domain          53/tcp          nameserver      # name-domain server
domain          53/udp          nameserver
mtp             57/tcp                          # deprecated
tftp            69/udp
rje             77/tcp          netrjs
finger          79/tcp
link            87/tcp          ttylink
supdup          95/tcp
hostnames       101/tcp         hostname        # usually from sri-nic
```

```
#csnet-cs       105/?
pop             109/tcp         postoffice
sunrpc          111/tcp
sunrpc          111/udp
auth            113/tcp         authentication
sftp            115/tcp
uucp-path       117/tcp
nntp            119/tcp         readnews untp   # USENET News Transfer Protocol
#
# UNIX specific services
#
exec            512/tcp
biff            512/udp         comsat
login           513/tcp
who             513/udp         whod
shell           514/tcp         cmd             # no passwords used
syslog          514/udp
printer         515/tcpspooler                  # line printer spooler
talk            517/udp
ntalk           518/udp
efs             520/tcp                         # for LucasFilm
route           520/udp         router routed
timed           525/udp         timeserver
tempo           526/tcp         newdate
courier         530/tcp         rpc
conference      531/tcp         chat
netnews         532/tcp         readnews
netwall         533/udp                         # -for emergency broadcasts
uucp            540/tcp         uucpd           # uucp daemon
remotefs        556/tcp        rfs_server rfs   # Brunhoff remote filesystem
ingreslock      1524/tcp
```

3.2.4 /etc/inetd.conf

When the **inetd** daemon starts it reads the */etc/inetd.conf* file to determine what services to invoke. Each line in the*/etc/inetd.conf* file provides **inetd** with the following information:

service Specifies the type of service, such as **ftp**, as found in the */etc/services* file.

endpoint type

Specifies the type of port that **inetd** expects to find associated with the service. The endpoint type can be stream or data gram.

protocol	Specifes the protocol type associated with the endpoint, either TCP or UDP.
wait/nowait	Specifies whether **inetd** should wait for one server to complete before starting another.
user name	Specifies the user name under which **inetd** runs.
name	Specifies the name of the server program.
argument	Specifies the argument passed to the server.

The */etc/inetd.conf* template file contains the following entries:

```
#
# etc/inetd.conf
#
# Internet server configuration database
#
ftp     stream  tcp     nowait  root    /etc/ftpd       ftpd
#telnet stream  tcp     nowait  root    /etc/telnetd    telnetd
shell   stream  tcp     nowait  root    /etc/reshd      reshd
login   stream  tcp     nowait  root    /etc/rlogind    rlogind
#exec   stream  tcp     nowait  root    /etc/rexecd     rexecd
#comsat dgram   udp     wait    root    /etc/comsat     comsat
#talk   dgram   udp     wait    root    /etc/talkd      talkd
#ntalk  dgram   udp     wait    root    /etc/ntalkd     ntalkd
#echo   stream  tcp     nowait  root    internal
#discard        stream  tcp     nowait  root    internal
#chargen        stream  tcp     nowait  root    internal
#daytime        stream  tcp     nowait  root    internal
time        stream  tcp     nowait  root    internal
#echo       dgram   udp     wait    root    internal
#discard    dgram   udp     wait    root    internal
#chargen    dgram   udp     wait    root    internal
#daytime    dgram   udp     wait    root    internal
#time       dgram   udp     wait    root    internal
tftp   dgram    udp    wait   nobody    /etc/tftpd     tftpd
```

Use the number (#) character to select or restrict the use of specific services.

3.2.5 /usr/hosts/*

A */usr/hosts/** directory can be created to allow users on one host to access, through **resh**, another host by simply typing the name of the remote system. For example, a user on *admin1* (Figure 2-1) could access *admin2* by typing `admin2` at the system prompt.

Host entries in the */usr/hosts/** directory are linked to the */usr/bin/resh* file.

3.3 Network Permissions Files

The network permissions files determine the level of access a user has to a remote system. The following files determine network permissions:

- */etc/hosts.equiv*
- *.rhosts*
- */etc/ftpusers*

The flowchart in Figure 3-1 uses two systems, *engr1* and *engr2*, (see Figure 2-1) to illustrate how the */etc/hosts.equiv* and *.rhosts* files control access to systems on a network.

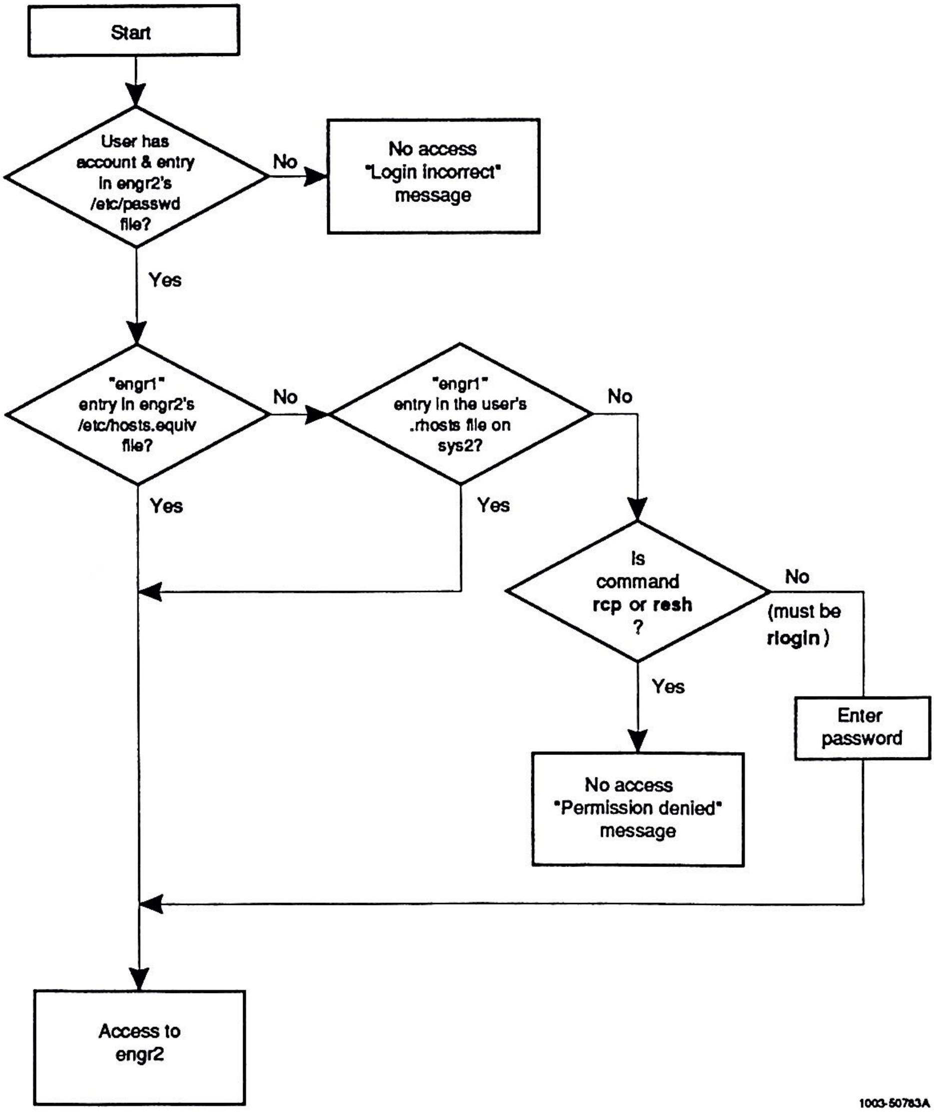

Start
User has account & entry in engr2's /etc/passwd file?
No
No access "Login incorrect" message
Yes
"engr1" entry in engr2's /etc/hosts.equiv file?
No
"engr1" entry in the user's .rhosts file on sys2?
No
Yes
Yes
Is command rcp or resh ?
No (must be rlogin)
Yes
Enter password
No access "Permission denied" message
Access to engr2
1003-50783A

Figure 3-1. Network permissions flowchart. The flowchart above shows the process used to determine if *engr1* can access *engr2*.

When a user on *engr1* attempts to access *engr2* through a **resh**, **rcp**, or **rlogin** command, *engr2* first checks to see if the user has an account on *engr2* and an entry in the */etc/passwd* file. With no account and no entry in the *engr2* password file, the *engr2* user is denied access to *engr2* and receives `login incorrect` message. If the user has an account on *engr2* and an entry in the */etc/passwd* file, *engr2* checks for an *engr1* entry in its */etc/hosts.equiv* file. With an entry in the */etc/hosts.equiv* file, the user is able to access *engr2*. When no *engr1* entry exists in the */etc/hosts.equiv* file, *engr2* checks for a *.rhosts* file with an *engr1* entry in the *engr1* account. If the *.rhosts* file with the correct entry exists, the *engr1* user is allowed access to *engr2*. When no *.rhosts* file exists, the user attempting to access *engr2* through the **rcp** or **resh** utilities is denied access and receives the `Permission denied` message. Users attempting access through the **rlogin** utility must enter the correct password.

3.3.1 /etc/hosts.equiv

The */etc/hosts.equiv* file allows users on one system to access another (remote) system using the **rlogin, resh**, and **rcp** utilities. The user does not need to supply a password when accessing the remote system if the following conditions are met:

- The remote system has an entry in its */etc/hosts.equiv* file for the user's system
- The user has an account (as indicated by an entry in the */etc/passwd* file) on the remote system

The */etc/hosts.equiv* file contains a list of all systems on the network that are considered logically equivalent to your local system. In other words, users on a system included in the */etc/hosts.equiv* file are dealt with exactly as those on the local system. This means that "trusted hosts" have the same privileges as the local system.

As shown below, the */etc/hosts.equiv* file for *engr1* in Figure 2-1 contains a single line entry for each system that has access to *engr1*:

```
engr2
engr3
mfg1
mfg2
mfg3
mfg4
```

3.3.2 .rhosts

The *.rhosts* file allows a subset of users on one host to access a remote host. This file allows an individual user to log in to a remote host without supplying a password when the remote host does not contain an entry in its */etc/hosts.equiv* file for the user's host. The user must first have an account on the remote system The user then needs to create an *.rhosts* file in their directory on the remote host.

The *.rhosts* file has the following format:

hostname userid

The *hostname* is the name of the user's local system and the *userid* is the user's login name. For example, a user on *engr1* (see Figure 2-1) with a login name of `fred` and an account on remote machine *engr2* might have the following *.rhosts* file on *engr2*:

```
engr1 fred
```

The login name on the local and remote system must be the same when no *userid* is specified.

3.3.3 /etc/ftpusers

The */etc/ftpusers* file specifies which users are *not* allowed to access a remote system using the **ftp** utility. The **ftp** server consults this file whenever it receives an **ftp** request from an **ftp** client process.

As shown in the following */etc/ftpuser* template file, each user name is entered on a single line:

```
uucp
ingres
root
```

3.4 Enabling rwhod

The **rwhod** server maintains the database used by the **rwho** and **ruptime** programs. Because of the network load that results from the broadcast messages generated by **rwhod**, it is by default not enabled.

To start **rwhod** when the system is rebooted, remove the comment characters (#) from the following lines in the */etc/init.d/netservers* file:

```
#
#if [ -f /etc/rwhod ]; then
#/etc/rwhod & echo ' rwhod#fi
#
```

To start the **rwhod** server manually, enter the following command at the super-user prompt:

```
/etc/rwhod
```

3.5 Network Utilities

The ptx/TCP/IP software has the following utilities:

- arp
- netstat
- ping
- route

Some of the these utilities are available to the general system user while others are intended for use by the system administrator.

3.5.1 arp

The **arp** command allows the administrator or user to display the contents of the logical-to-physical address resolution table. However, only the network administrator can enter, delete or change the entries in the table.

To display the current ARP table enter the following command:

```
arp –a [in_addr]
```

in_addr is the Internet Address of the entry of a specific table entry.

If you do not supply the optional Internet Address, the complete table is displayed as shown below:

```
Internet Addresses    Ethernet Addresses
89.0.33.15            08.00.39.00.21.70
89.0.3.14             08.00.4E.01.1E.85
89.0.20.16            08.00.39.00.20.96
89.0.21.16            08.00.39.00.21.F1
89.0.20.144           08.00.39.00.20.49
89.0.21.15            08.00.2b.06.6B.2B
```

If there is more than one Ethernet interface, for example when a system is used as a gateway, the arp table for all the interfaces is displayed.

You can insert an entry, consisting of a hostname and the corresponding Internet and physical addresses, in the table with the **–s** option as shown below:

arp –s *in.addr eth.addr*

eth.addr is the physical address, for example, the Ethernet Address, which you wish to enter. This is useful for hosts which do not support ARP. Entries added by this option remain in the table until deleted by the super user. Entries containing addresses received on the network are removed if unused for 10 minutes.

To delete a host use the –d option as shown in the following command;

arp –d *in.addr*

The Internet Address should be entered in hexadecimal form using dotted notation (separating the fields by dots). If you do not specify an Internet Address in the command line, you will be prompted for one after pressing Return.

3.5.2 netstat

The **netstat** utility allows users to display the status of the network. The **netstat** command has the following two command forms:

netstat [**–an**] [**–a**]
netstat [**–imnrs**]

There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of protocol control blocks for each protocol. The second form presents the contents of one of the other network data structures according to the option selected.

The options have the following meaning:

–a	Show the state of all protocol control blocks; normally protocol control blocks used by server processes are not shown.
–i	Show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown).
–m	Show statistics recorded by the memory management routines (the network manages a private share of memory).
–n	Show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically).
–s	Show per-protocol statistics
–r	Show the routing tables. When **–s** is also present, show routing statistics instead.

There are a number of display formats, depending on the information presented. The default display, for active protocol control blocks, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol.

Address formats are of the form *host.port* or *network.port* if a protocol control block's address specifies a network but no specific host address. When known, the host and network addresses are displayed symbolically according to the data bases */etc/hosts* and */etc/networks*, respectively or through the name server. If a symbolic name for an address is unknown, or if the **–n** option is specified, the address is printed in the Internet dot format. Unspecified, or wildcard, addresses and ports appear as a star (*).

The interface display provides a table of cumulative statistics regarding packets transferred and packets dropped on output. The network address (currently Internet specific) of the interface and the maximum transmission unit (mtu) are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route (U if up), whether the route is to a gateway (G), and whether the route was created dynamically by a redirect (D). Direct routes are created for each interface attached to the local host. The ttl field indicates the remaining time-to-live for this route. PERM routers are created via the route command and are never timed out. A route begins aging after each reference. Each reference to the route resets the timer to the maximum value. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

3.5.3 ping

The **ping** utility sends the ICMP ECHO_REQUEST packets to network hosts. The **ping** command has the following syntax:

```
/etc/ping [ -v ] host [ count ] [ packetsize ]
```

Ping uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams (pings) have an IP and ICMP header, followed by a time value and then an arbitrary number of *pad* bytes used to fill out the packet. The default datagram length is 64 bytes, but this may be changed using the command-line option.

The **–v** option provides verbose output. This option lists received ICMP packets other than ECHO RESPONSE packets.

When using *ping* for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be "pinged." **Ping** sends one datagram and prints one line of output. If an optional *count* is given, that number of requests is sent, and one line for every ECHO_RESPONSE returned is printed. No output is produced if there is no response. Round-trip times and packet loss statistics are computed. When all responses have been received or the program times out (*count* packets sent), or if the program is terminated with a SIGINT, a brief summary is displayed.

This program is intended for use in network testing, measurement and management. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is unwise to use **ping** during normal operations or from automated scripts.

3.5.4 route

The **route** utility allows the administrator to manually add or delete routing table entries. Only the superuser may modify the routing table with the **route** command. The **route** command has the following syntax:

/etc/route [**–f**] [**–n**] *command* [**net** | **host**] **destination gateway** [*metric*]

The *command* options are either **add** or **delete**. The **add** command adds a new network-gateway pair to the routing table. The **delete** command removes one or more network-gateway pairs from the routing table.

Using the **–f** option causes the **route** command to "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application. The **–n** option prevents attempts to print host and network names symbolically when reporting actions.

The **destination** option is a host or network, **gateway** is the next-hop gateway to which packets should be addressed, and *metric* is a count indicating the number of hops to the **destination**. Both **destination** and **gateway** can be expressed as names or Internet addresses. If expressed as names, *destination* must be a valid entry in the */etc/networks* file and **gateway** must be a valid entry in the */etc/hosts* file. The **add** command requires a *metric* count. The *metric* count must be zero if the destination is on a directly-attached network, and nonzero if the route utilizes one or more gateways. If adding a route with metric 0, the gateway given is the address of the gateway host on the common network, indicating the interface to be used for transmission. The optional keywords **net** and **host** force the **destination** to be interpreted as a network or a host, respectively. If the route is to a destination connected by a gateway, the *metric* should be greater than 0. All addresses must be specified in dotted-decimal form.

thero
cisco
slip

4. User Datagram Protocol

4. User Datagram Protocol

Chapter 4

Domain Name Server Configuration

4.1 Introduction 4-1

4.2 Overview 4-1

4.3 Types of Servers 4-2
- Master Servers 4-2
- Caching Servers 4-2
- Remote Servers 4-2

4.4 Name Server Files 4-3
- named.boot 4-4
- resolv.conf 4-4
- named.ca 4-5
- named.local 4-5
- named.hosts 4-5
- named.hosts.rev 4-6

4.5 Preparing Your System for the Name Server 4-6

4.6 Starting the Name Server 4-7

4.7 Sample Name Server Configuration 4-7
  - /etc/named.boot 4-8
  - /etc/named.local 4-8
  - /etc/resolv.conf 4-8
  - /etc/named.hosts 4-9
  - /etc/named.hosts.rev 4-9

Tables

4-1 **named** Server Files 4-3

Chapter 4

Domain Name Server Configuration

4.1 Introduction

This chapter describes the Berkeley Internet Name Domain (BIND). facility including the files that must be edited to configure the name server on your system. The information in this chapter is taken from the *Name Server Operations Guide for BIND* by Kevin J. Dunlap. Appendix B, "Name Server Operations Guide," contains the complete text of Dunlap's guide.

4.2 Overview

The BIND facility is the Internet name server implementation for Unix systems. With a name server, the host name and address data base for an entire network can be divided among several hosts, thus eliminating the need for a site to maintain a large data base such as the */etc/hosts* file. This is especially important for sites attached to the Internet, where it is impractical for a single host to maintain the Network Information Center's (NIC) entire global host name and address resolution table.

With a name server, a network can be divided into groups known as *domains*. A domain's name server program only handles queries about hosts within its domain. A query about a host in a different domain would be sent to that host's domain name server. Domains can be further divided into subdomains with each subdomain having its own name server.

BIND consists of a user interface called a **resolver** and a name server daemon called **named**. The **resolver** consists of routines that query **named** for information about other devices on the network. These routines have been integrated into the **gethostbyname** and **gethostbyaddr** routines that reside in */usr/lib/libinet.a*.

The addition of the resolver routines changes the way the **gethostbyname** and **gethostbyaddr** routines function. In particular, the **gethostbyname** routine first checks for the existence of a name server. If no name server is present, **gethostbyname** looks at the */etc/hosts* file for network information.

The **named** server listens on a well-known port for resolver queries about network services. Most commonly, **named** provides the hostname-to-address service required by most network applications. It can also provide data on a variety of network objects including mailboxes and printers.

4.3 Types of Servers

The **named** process specifies three types of servers: master, caching, and remote. The following sections describe each server.

4.3.1 Master Servers

The master server is the name authority for a domain. It maintains all the data corresponding to its domain or zone of authority. A Master server can be either a *primary master server* or a *secondary master server*.

The primary master server receives its data from a file on disk. It can also delegate authority to another server. The secondary master server receives its delegated authority from the primary master server. When the secondary master server boots, it requests all the domain data from the primary master server. It also checks periodically with the primary master server to determine if it needs to update its data.

A server may be a master for more than one domain. It can be a primary for one domain and a secondary for another domain.

4.3.2 Caching Servers

Caching servers temporarily store network information and provide it when queried. All servers are caching servers. A specific type of caching server, known as a *caching only server*, responds to queries but must ask other servers with the proper authority for the required information.

4.3.3 Remote Servers

Remote servers, such as workstations, provide the features of a name server without the **named** process actually running on the machine. When queried, a remote server must use the name server running on another machine.

4.4 Name Server Files

The **named** server uses the following files:

- *named.boot*
- *resolv.conf*
- *named.ca*
- *named.local*
- *named.hosts*
- *named.hosts.rev*

Each of the files listed above is one of two types, boot or domain data. Table 4-1 identifies the file type and lists the **named** files that run on each server.

Table 4-1
named Server Files

File Type	File	Server Location
Boot	*named.boot*	master and caching servers
	resolve.conf	master, caching and remote
	named.ca	master and caching servers
Domain Data	*named.local*	master and caching servers
	named.hosts	master servers
	named.hosts.rev	master servers

The following sections describes each of the **named** files and shows a template of each file. The ptx/TCP/IP software contains the **named** template files in the */etc* directory. These templates can be used to create the working **named** files for your system.

4.4.1 named.boot

The *named.boot* file is the first file read when **named** starts. This file tells the server the following information:

- The type of **named** server it is (primary master, secondary master, or caching).
- Where to get its initial data
- Its zone of authority

The */etc* directory is usually the default location for *named.boot*. All hosts running **named** must have a *named.boot* boot file.

The ptx/TCP/IP software contains the following **named.boot** template file:

```
;
;  boot file for authoritive master name server for COMPANYNAME.COM
;  Note that there should be one primary entry for each SOA record.
;
;

; type          domain                    source host/file

domain          COMPANYNAME.com
primary         COMPANYNAME.COM           /etc/named.hosts
primary         0.103.IN-ADDR.ARPA        /etc/named.hosts.rev
primary         0.0.127.IN-ADDR.ARPA      /etc/named.local
```

4.4.2 resolv.conf

The *resolv.conf* file designates the name server that should be sent queries. All servers in a domain require the resolv.conf file. The ptx/TCP/IP software contains the following *resolv.conf* template file:

```
#domain        sqnt.com
#nameserver     127.1
```

4.4.3 named.ca

The *named.ca* file contains information that a name server needs to know about the root authoritative servers for a network. The ptx/TCP/IP software does not contain a *named.ca* template file. Refer to Appendix B, Section B.5.8 for an example of a *named.ca* file.

4.4.4 named.local

The *named.local* file specifies the address for the local loopback interface (**localhost**) as 127.0.0.1. All hosts running **named** must have this file. The *named.boot* file specifies the location of *named.local*.

The ptx/TCP/IP software contains the following *named.local* template file:

```
;
;   named.local
;

@    IN     SOA       MACHINE.COMPANYNAME.com  root.MACHINE.COMPANYNAME.com (
                      1.1     ;Serial
                      3600    ;Refresh
                      300     ;Retry
                      3600000 ;Expire
                      14400 ) ;Minimum
     IN     NS        MACHINE.COMPANYNAME.com.
1    IN     PTR       localhost.
```

4.4.5 named.hosts

The *named.hosts* file contains the host name and address information for all systems in a specific zone of authority. The *named.boot* file specifies the location of *named.hosts*.

The ptx/TCP/IP software contains the following *named.hosts* template file:

```
;
; Authoritative data for SOMECOMPANY.COM (ORIGIN assumed SOMECOMPAY.COM)
;
@           IN     SOA    MACHINE.COMPANYNAME.com root.MACHINE.COMPANYNAME.com (
                          1.1           ; Serial
                          10800         ; Refresh 3 hours
                          3600          ; Retry   1 hour
                          3600000       ; Expire  1000 hours
                          86400 )       ; Minimum 24 hours
localhost   IN     A      127.1
HOST_A      IN     A      103.0.1.03
            IN     HINFO  s27
```

```
HOST_B          IN      A        103.0.1.06
                IN      HINFO    s27
HOST_C          IN      A        103.0.1.07
                IN      HINFO    s81
```

4.4.6 named.hosts.rev

The *named.hosts.rev* file specifies the IN-ADDR.ARPA domain. The IN-ADDR.ARPA domain allows address to name mapping. As internet host addresses do not fall within domain boundaries, this special domain allows inverse mapping.

Four labels precede the IN-ADDR.ARPA domain. These labels correspond to the 4 octets of an Internet address. For example, the Internet address 128.32.0.4 is located in the domain 4.0.32.128.IN-ADDR.ARPA. All four octets must be specified even if an octet is zero. This reversal of the address is awkward to read but allows for the natural grouping of hosts in a network.

The ptx/TCP/IP software contains the following *named.hosts.rev* template file:

```
;
;   sample reverse address example
;

@   IN  SOA MACHINE.COMPANYNAME.COM root.MACHINE.COMPANYNAME.com (
                1.1     ; Serial
                10800   ; Refresh 3 hours
                3600    ; Retry   1 hour
                3600000 ; Expire  1000 hours
                86400 )    ; Minimum 24 hours
        IN  NS  HOST_A.COMPANYNAME.COM
3.1     IN  PTR HOST_B.COMPANYNAME.COM.
6.1     IN  PTR HOST_C.COMPANYNAME.COM.
7.1     IN  PTR HOST_A.COMPANYNAME.COM.
```

4.5 Preparing Your System for the Name Server

Prior to setting up the **named** server on your system, you should consider the following points:

- Is the network already running the **named** server?
- What type of server will your system be?

If you are setting up your system on a network that is already running the **named** server, you will configure your system with the existing domain name. Networks not currently running **named** will need to obtain a domain name. Section B.4, "Setting Up Your Domain" describes how to obtain domain names for sites that will be attached to public networks. You can choose your own domain name if you will not be attaching your network to a public network.

You will also need to determine what type of server, master, caching, or remote, your system will be. Selection of a server type depends upon how you layout of network in regards to domains, subdomains, and zones of authority.

4.6 Starting the Name Server

By default the name server is not started. Once you have edited the name server files, you can start the name server manually by entering the following command at the super user prompt:

```
# /etc/named
```

You can also start the name server when the system is rebooted by removing the comment characters (#) from the following lines in the */etc/init.d/netservers* file:

```
#
#if [ -f /etc/named ]; then
#/etc/named & echo ' named#fi
#
```

4.7 Sample Name Server Configuration

This section provides a sample name server set up based on the network shown in Figure 2-1. To illustrate how each **named** file should be edited *engr2* has been designated as a Primary Master Server. The name server configuration worksheet (Appendix A) for *engr2* is shown below. The sample company name ACE has been selected to illustrate the proper format for commercial (COM) networks.

Type of Server (Master, Caching, or Remote) __master____

Type of Master Server (Primary or Secondary) __primary___

Default Domain Name __ace.com__

Source File __/etc/named.hosts__

Primary Name Server Network Address __102.0.0.2___

/etc/named.boot

The */etc/named.boot* file for *engr2* would be edited as shown below:

```
;
;  boot file for authoritive master name server for ACE.COM
;  Note that there should be one primary entry for each SOA record.
;
;

; type          domain                      source host/file

domain          ace.com
primary         ACE.COM                     /etc/named.hosts
primary         0.102.IN-ADDR.ARPA          /etc/named.hosts.rev
primary         0.0.127.IN-ADDR.ARPA        /etc/named.local
```

/etc/named.local

The */etc/named.local* file for *engr2* would be edited as shown below:

```
;
;   named.local
;

@     IN     SOA     engr2.ace.com          root.engr2.ace.com (
                     1.1      ;Serial
                     3600     ;Refresh
                     300      ;Retry
                     3600000 ;Expire
                     14400 ) ;Minimum
      IN     NS      engr2.ace.com.
1     IN     PTR     localhost.
```

/etc/resolv.conf

The */etc/resolv.conf* file for *engr2* would be edited as shown below. Note that the comment character (#) at the beginning of each line has been removed.

```
domain          ace.com
nameserver       102.0.0.2
```

/etc/named.hosts

The */etc/named.hosts* file for *engr2* would be edited as shown below:

```
;
; Authoritative data for ACE.COM (ORIGIN assumed ACE.COM)
;
@           IN     SOA    engr2.ace.com root.engr2.ace.com (
                          1.1           ; Serial
                          10800         ; Refresh 3 hours
                          3600          ; Retry   1 hour
                          3600000       ; Expire  1000 hours
                          86400 )       ; Minimum 24 hours
             IN    NS     engr2.ace.com
 localhost   IN    A      127.1
 engr1       IN    A      102.0.0.1
             IN    HINFO  s3
 engr3       IN    A      102.0.0.3
             IN    HINFO  s3
 mfg3        IN    A      102.0.0.4
             IN    HINFO  s27
 mfg1        IN    A      101.2.0.1
             IN    HINFO  s27
 mfg2        IN    A      101.2.0.2
             IN    HINFO  s81
 mfg4        IN    A      101.2.0.4
             IN    HINFO  s16
 admin1      IN    A      101.1.0.1
             IN    HINFO  s27
 admin2      IN    A      101.1.0.2
             IN    HINFO  s3
 admin3      IN    A      101.1.0.3
             IN    HINFO  s3
```

/etc/named.hosts.rev

The */etc/named.hosts.rev* file for *engr2* would be edited as shown below:

```
;
;   named.hosts.rev file for engr2
;

@           IN     SOA    engr2.ace.com root.engr2.ace.com (
                          1.1           ; Serial
                          10800         ; Refresh 3 hours
                          3600          ; Retry   1 hour
                          3600000       ; Expire  1000 hours
                          86400 )       ; Minimum 24 hours
               IN      NS       engr2.ace.com
 1.0.0.102     IN      PTR      engr1
 3.0.0.102     IN      PTR      engr3
```

```
4.0.0.102      IN       PTR       mfg3
1.0.2.101      IN       PTR       mfg1
2.0.2.101      IN       PTR       mfg2
4.0.2.101      IN       PTR       mfg4
1.0.1.101      IN       PTR       admin1
2.0.1.101      IN       PTR       admin2
3.0.1.101      IN       PTR       admin3
```

5. Sendmail Configuration

5. Sendmail Configuration

Chapter 5
Sendmail Configuration

5.1 Introduction 5-1
5.2 Sendmail Overview 5-1
5.3 Setting Up Sendmail 5-3

Figures

5-1 Sendmail structure 5-1

Chapter 5
Sendmail Configuration

5.1 Introduction

This chapter describes how to configure your system to run the Sendmail facility. Understanding all aspects of the Sendmail facility can be a daunting task. Therefore, the approach taken in this chapter is to provide a brief overview along with a basic procedure for implementing Sendmail.

For a detailed description of Sendmail, including how to write your own configuration file, refer to the following appendixes:

- Appendix C, "Sendmail Overview"
- Appendix D, "Sendmail Installation and Operation Guide"

5.2 Sendmail Overview

Sendmail is a general purpose UNIX mail routing facility. Through Sendmail, a message can be sent to one or more recipients. The sender and recipient can be on the same machine or separate machines on different types of networks. For example, Sendmail can send messages over both UUCP networks and networks using the TCP/IP transport protocol.

It is important to note that Sendmail is neither a mail-user interface or a mail delivery program. Rather, as shown in Figure 5-1, Sendmail acts as a router between a specific mail-sending program such as *Mail* and a mail delivery (mailer) program such as */bin/mail* or */bin/rmail*. The user interacts with the mail generation and sending program to create a mail message. After the message has been created, the mail generation program calls Sendmail to route the message to the correct mailer.

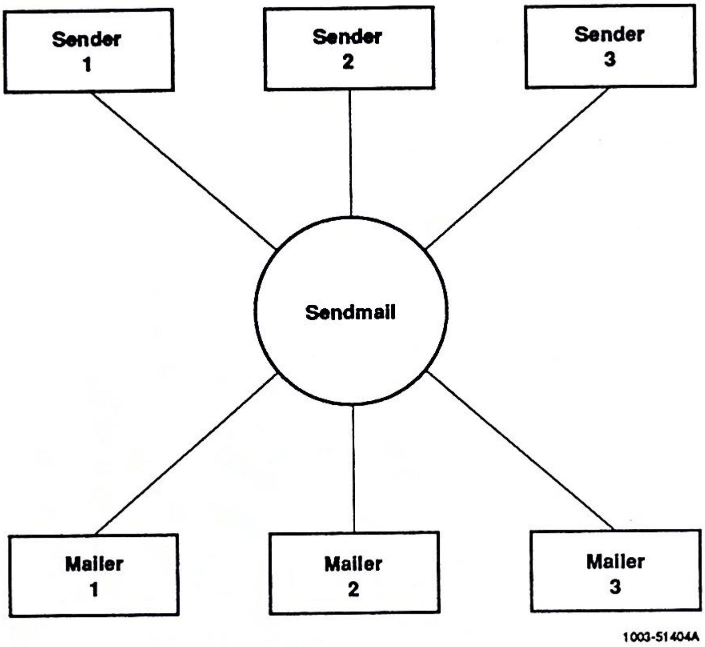


Figure 5-1. Sendmail structure. Sendmail acts as a switch to route mail from a sending program to a mailer program.

The Sendmail facility operates in two distinct phases. During the first phase, it collects and stores the sender's message. During the second phase, it examines the sender's address and the address of the receiver. It changes the addresses as required and delivers the message to the proper mailer. If the mailer cannot handle the message immediately, Sendmail queues the message for later transmission to the mailer. If any errors occur during the second phase, Sendmail either creates and returns a new message describing the error or returns a status code telling what went wrong.

At the heart of the Sendmail facility is the complex configuration file, *sendmail.cf*. The *sendmail.cf* file contains macros and rewriting rulesets that change address formats and headers to accommodate the various mailers.

The ptx/TCP/IP software contains the following off-the-shelf Sendmail configuration files that you can modify for use at your site:

tcpproto.cf	Used on systems connected by a network that uses TCP/IP as the communication protocol.
uucpproto.cf	Used on systems connected by UUCP only.
tcpuucpproto.cf	Used on a system that connects to other systems through a TCP/IP network and *also* connects to the outside world through UUCP.

5.3 Setting Up Sendmail

This section describes how to do a basic Sendmail setup. The */usr/lib/sendmail.cf* file that is installed as part of the ptx/TCP/IP software will work without modification for most simple sites that do not connect to the outside world through UUCP. This file also works for systems running the domain name server or using host tables (*/etc/hosts*). For more complex Sendmail configurations, refer to the "Sendmail Installation and Operation Guide" in Appendix D.

The following sequence describes how to start the Sendmail facility:

1. Execute either the **newaliases** command or the following **sendmail** command at the superuser prompt to create the DBM version of the alias database:

   ```
   /usr/lib/sendmail -bi
   ```

2. **Start the Sendmail facility.** Change the system run level to the multiuser mode using the Change System Run Level menu. Sendmail can be started manually by executing the following command at the superuser prompt:

   ```
   /usr/lib/sendmail -bd -q30m
   ```

   The `-q30m` flag specifies that the mail queue should be processed every 30 minutes.

3. **Test Sendmail on your system.** Execute the following command with a known user to test the Sendmail facility:

   ```
   /usr/lib/sendmail -bv user
   ```

   For example, a Sendmail facility that is operating correctly will return the following message when the above command is typed for the user `john`.

   ```
   john... deliverable
   ```

A. Configuration
Worksheets

A. Configuration Worksheets

Appendix A
Configuration Worksheets

A.1 Introduction A-1

A.2 Base Network Software Worksheet A-1

A.3 Domain Name Server Worksheet A-3

Appendix A
Configuration Worksheets

A.1 Introduction

This appendix provides worksheets that assist you in the installation of the ptx/TCP/IP software. These worksheets cover installation of the base software and the Berkeley Internet Name Domain (BIND) Server. Each worksheet should be completed before attempting to install the ptx/TCP/IP software.

A.2 Base Network Software Worksheet

The following base network software worksheet lists the information required to edit the base network files, */etc/netconf* and */etc/hosts*. The following list describes the information required for each worksheet entry:

System Name	The name you have selected for your system using the `Set System Node Name` menu. For example, the system names for the hosts shown in Figure 2-1 include `admin1` and `mfg4`.
Prim Address	The primary internet address assigned to your host.
Interface	The SCED-based or SSM-based interface hardware used to attach to the network. Interface entries for SCED-based systems are `se0`, or `se1`. SSM-based systems are indicated as `eg0`, or `eg1`.
Subnet Mask	This value replaces the default net mask used for the internet address. It may be expressed in either dotted decimal from (example, 255.255.0.0) or hexadecimal (example, 0xffff0000).
IP Brdcst	The number 1 or 0 indicating a broadcast address of all 1's or all 0's.

Default Gateway	The IP address of the system that acts as the gateway to other networks.
IP Forwarding	Indicates if packets should be forwarded.
Name Server	Indicates if the name server is used.

System Name ____________

Prim Addr______________ Interface____ Subnet Mask__________ IP Brdcst____

IP Addr #1______________ Interface____ Subnet Mask__________ IP Brdcst____

IP Addr #2______________ Interface____ Subnet Mask__________ IP Brdcst____

IP Addr #3______________ Interface____ Subnet Mask__________ IP Brdcst____

Default Gateway___________

IP Forwarding [on/off]___________

Name server [on/off]___________

A.3 Domain Name Server Worksheet

The following domain name server worksheet lists the information required to setup your system with the **named** server. Refer to Chapter 4, "Domain Name Server Configuration" and Appendix B, "Name Server Operations Guide" for a description of the parameters listed below.

Type of Server (Master, Caching, or Remote) ______________

Type of Master Server (Primary or Secondary) ______________

Default Domain Name ______________

Boot File (for **named**)______________

Primary Name Server Network Address ______________

B. Name Server
Operations Guide

B. Name Server
Operations Guide

Appendix B
Name Server Operations Guide

B.1 Introduction B-1

B.2 Building A System with a Name Server B-1
Resolver Routines in libc B-1
The Name Service B-2

B.3 Types of Servers B-3
Master Servers B-3
Primary B-3
Secondary B-3
Caching Only Server B-3
Remote Server B-4

B.4 Setting up Your Own Domain B-4
DARPA Internet B-4
CSNET B-4
BITNET B-5

B.5 Files B-5
Boot File B-5
Domain B-5
Sortlist B-5
Primary Master B-6
Secondary Master B-6
Caching Only Server B-6
Remote Server B-6
Cache Initialization B-7
named.ca B-7
Domain Data Files B-7
named.local B-7
hosts B-7
hosts.rev B-7
Standard Resource Record Format B-8
$INCLUDE B-9
$ORIGIN B-9
SOA - Start Of Authority B-9
NS - Name Server B-10

    A - Address B-10
    HINFO - Host Information B-10
    WKS - Well Known Services B-10
    CNAME - Canonical Name B-11
    PTR - Domain Name Pointer B-11
    MB - Mailbox B-11
    MR - Mail Rename Name B-11
    MINFO - Mailbox Information B-12
    MG - Mail Group Member B-12
    MX - Mail Exchanger B-12
Sample Files B-13
Boot File B-13
    Primary Master Server B-13
Remote Server B-14
    /etc/resolv.conf B-14
named.ca B-15
named.local B-15
UCB Hosts B-16
UCB Host.rev B-18

B.6 Domain Management B-18
/etc/rc.local B-18
/etc/named.pid B-19
/etc/hosts B-19
Signals B-19
    Reload B-19
    Debugging B-19

Appendix B
Name Server Operations Guide

This appendix provides detailed information on the Berkeley Internet Name Domain (BIND). It is taken from the *Name Server Operations Guide for BIND* by Kevin J. Dunlap.

B.1 Introduction

The Berkeley Internet Name Domain (BIND) Server implements the DARPA Internet name server for the UNIX* operating system. A name server is a network service that enables clients to name resources or objects and share this information with other objects in the network. This in effect is a distributed data base system for objects in a computer network. BIND is fully intergrated into 4.3BSD network programs for use in storing and retrieving host names and address. The system administrator can configure the system to use BIND as a replacement to the original host table lookup of information in the network hosts file */etc/hosts*. The default configuration for 4.3BSD uses BIND.

B.2 Building A System with a Name Server

BIND is comprised of two parts. One is the user interface called the **resolver** which consists of a group of routines that reside in the C library */usr/lib/libinet.a*. Second is the actual server called **named**. This is a daemon that runs in the background and services queries on a given network port. The */etc/services* file specifies the standard ports for UDP and TCP.

B.2.1 Resolver Routines in libc

When building your 4.3BSD system you may either build the C library to use the name server resolver routines or use the host table lookup routines to do host name and address resolution. The default resolver for 4.3BSD uses the name server.

---

* UNIX is a Trademark of AT&T Bell Laboratories

Building the C library to use the name server changes the way **gethostbyname**, **gethostbyaddr**, and **sethostent** do their functions. The name server renders **gethostent** obsolete, since it has no concept of a next line in the database. These library calls are built with the resolver routines needed to query the name server.

The **resolver** is comprised of a few routines that build query packets and exchange them with the name server.

B.2.2 The Name Service

The basic function of the name server is to provide information about network objects by answering queries. The specifications for this name server are defined in RFC882, RFC883, RFC973, and RFC974. These documents can be found in */usr/src/etc/named/doc* in 4.3BSD or copied using **ftp** from sri-nic.arpa. It is also recommeded that you read the related manual pages, **named**, **resolver**, and **resolver**.

The advantage of using a name server over the host table lookup for host name resolution is to avoid the need for a single centralized clearinghouse for all names. The authority for this information can be delegated to the different organizations on the network responsible for it.

The host table lookup routines require that the master file for the entire network be maintained at a central location by a few people. This works fine for small networks where there are only a few machines and the different organizations responsible for them cooperate. But this does not work well for large networks where machines cross organizational boundaries.

With the name server, the network can be 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.

B.3 Types of Servers

There are three types of servers: master, caching and remote.

B.3.1 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.

B.3.2 Caching Only Server

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, who have the authority, for the information needed. All servers keep data in their cache until the data expires, based on a time to live field attached to the data when it is received from another server.

B.3.3 Remote Server

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 the name server running on the local machine. All of the queries are serviced by a name server that is running on another machine on the network.

B.4 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. An organization that belongs to multiple networks (such as CSNET, DARPA Internet and BITNET) should register with only one network.

B.4.1 DARPA Internet

Sites that are already on the DARPA Internet and need information on setting up a domain should contact hostmaster@sri-nic.arpa. You may also want to be placed on the BIND mailing list, which is a mail group for people on the DARPA Internet running BIND. The group discusses future design decisions, operational problems, and other related topic. The address to request being placed on this mailing list is:

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

B.4.2 CSNET

A CSNET member organization that has not registered its domain name should contact the CSNET Coordination and Information Center (CIC) for an application and information about setting up a domain.

An organization that already has a registered domain name should keep the CIC informed about how it would like its mail routed. In general, the CSNET relay will prefer to send mail via CSNET (as opposed to BITNET or the Internet) if possible. For an organization on multiple networks, this may not always be the preferred behavior. The CIC can be reached via electronic mail at *cic@sh.cs.net*, or by phone at (617) 497-2777.

B.4.3 BITNET

If you are on the BITNET and need to set up a domain, contact the following:

```
INFO@BITNIC.
```

B.5 Files

The name server uses several files to load its data base. This section covers the files and their formats needed for **named**.

B.5.1 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 when you compile **named** or by specifying the location on the command line when **named** is started up.

Domain

The line in the boot file that designates the default domain for the server looks as follows:

```
domain          Berkeley.Edu
```

The name server uses this information when it receives a query for a name without a dot (.). When it receives one of these queries, it appends the name in the second field to the query name.

Sortlist

The line in the boot file that designates a preference order for sorting address looks as follows:

```
sortlist            10.0.0      128.45.0.0
```

This line is only needed if you want to control the order that addresses are returned from the name server. This is useful in cases where you have a point-to-point link to another network and you want the address using the link to take preference over going over another network.

Primary Master

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

```
primary         Berkeley.Edu    /etc/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 the primary except for the word secondary and the third field.

```
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 rest of the line, lists the network addresses for 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.

Caching Only Server

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

All servers should have a line as follows in the boot file to prime the name servers cache:

```
cache           .       /etc/named.ca
```

For information on cache file see section on *Cache Initialization*.

Remote Server

To set up a host that will use a remote server instead of a local server to answer queries, you need to create the file */etc/resolv.conf*. This file designates the name servers on the network that should be sent queries. It is not advisable to create this file if you have a local server running. If this file exists it is read almost every time **gethostbyname** or **gethostbyaddr** is called.

B.5.2 Cache Initialization

named.ca

The name server needs to know the server that is the authoritative name server for the network. To do this we have to prime the name server's cache with the address of these higher authorities. This file should only contain the information pertaining to the root authoritative servers for the network. The location of this file is specified in the boot file. This file uses the Standard Resource Record Format covered further on in this paper.

B.5.3 Domain Data Files

There are three standard files for specifying the data for a domain. These are *named.local*, *hosts*, and *host.rev*. These files use the Standard Resource Record Format covered later in this paper.

named.local

This 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.

hosts

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

hosts.rev

This file specifies the IN-ADDR.ARPA domain. This is a special domain for allowing address to name mapping. As 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 4 octets of an Internet address. All four octets must be specified even if an octets 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.

B.5.4 Standard Resource Record Format

The records in the name server data files are called resource records. The Standard Resource Record Format (RR) is specified in RFC882 and RFC973. The following is a general description of these records:

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

Resource records have a standard format shown above. The first field is always the name of the domain record. For some RR's the name may be left blank; in that case it takes on the name of the previous RR. The second field is an optional time to live field. This specifies how long this data will be stored in the data base. By leaving this field blank the default time to live is specified in the `Start Of Authority` resource record (see below). 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 RR. Case is preserved in names and data fields when loaded into the name server. All comparisons and lookups in the name server data base 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.
..	Two free standing dots represent the null domain name of the root when used in the name field.
\X	Where X is any character other than a digit (0-9), 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	Where each D is a digit, 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.
;	Semicolon starts a comment; the remainder of the line is ignored.
*	An asterisk signifies wildcarding.

Most resource records will have the current origin appended to names if they are not terminated by a dot (.). This is useful for appending the current domain name to the data, such as machine names, but may cause problems where you do not want this to happen. A good rule of thumb is that, if the name is not in of the domain for which you are creating the data file, end the name with a dot (.).

$INCLUDE

An include line begins with **$INCLUDE**, starting in column 1, and is followed by a file name. This feature is particularly useful for separating different types of data into multiple files. An example would be:

```
$INCLUDE /usr/named/data/mailboxs
```

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.

$ORIGIN

The origin is a way of changing 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.

SOA - Start Of Authority

```
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 Start of Authority, `SOA`, record designates the start of a zone. The name is the name of the zone. Origin is the name of the host on which this data file resides. Person in charge is the mailing address for the person responsible for the name server. The serial number is the version number of this data file, this number should be incremented whenever a change is made to the data. The name server cannot handle numbers over 9999 after the decimal point. The refresh indicates how often, in seconds, a secondary name servers is to check with the primary name server to see if an update is needed. The retry indicates how long, in seconds, a secondary server is to retry after a failure to check for a refresh. Expire is the upper limit, in seconds, that a secondary name server is to use the data before it expires for lack of getting a refresh. Minimum is the 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.

NS - Name Server

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

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 Primary Master server for the domain.

A - Address

```
{name}  {ttl}    addr-class      A        address
ucbarpa          IN              A        128.32.10.4
                 IN              A        10.0.0.78
```

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.

HINFO - Host Information

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

Host Information resource record, `HINFO`, is for host specific data. This 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 info and the operating system info. 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.

WKS - Well Known Services

```
{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 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 come from the list of services specified in */etc/services*. There should be only one `WKS` record per protocol per address.

CNAME - Canonical Name

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

Canonical Name resource record, `CNAME`, specifies an alias for a canonical name. An alias should be unique and all other resource records should be associated with the canonical name and not with the alias. Do not create an alias and then use it in other resource records.

PTR - Domain Name Pointer

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

A Domain Name Pointer record, `PTR`, allows special names to point to some other location in the domain. The above 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 *hosts.rev* file. `PTR` names should be unique to the zone.

MB - Mailbox

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

`MB` is the Mailbox record. This lists the machine where a user wants to receive mail. The name field is the users login; the machine field denotes the machine to which mail is to be delivered. Mail Box names should be unique to the zone.

MR - Mail Rename Name

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

Main Rename, `MR`, can be used to list 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.

MINFO - Mailbox Information

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

Mail Information record, `MINFO`, creates a mail group for a mailing list. This resource record is usually associated with a mail group *Mail Group*, but may 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.

MG - Mail Group Member

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

Mail Group, `MG` lists members of a mail group.

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.
```

MX - Mail Exchanger

```
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.
```

Main Exchanger records, `MX`, are used to specify 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 can not 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. 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.

B.5.5 Sample Files

The following section contains sample files for the name server. This covers example boot files for the different types of servers and example domain data base files.

B.5.6 Boot File

Primary Master Server

```
;
; Boot file for Primary Master Name Server
;


; type      domain                  source file or host
;
domain      Berkeley.Edu
primary     Berkeley.Edu            /etc/ucbhosts
cache       .                       /etc/named.ca
primary     32.128.in-addr.arpa     /etc/ucbhosts.rev
primary     0.0.127.in-addr.arpa    /etc/named.local
```

Secondary Master Server.

```
;
```

```
; Boot file for Secondary Master Name Server
;


; type        domain                  source file or host
;
domain        Berkeley.Edu
secondary     Berkeley.Edu            128.32.0.4 128.32.0.10 128.32.136.22
cache         .                       /etc/named.ca
secondary     32.128.in-addr.arpa     128.32.0.4 128.32.0.10 128.32.136.22
primary       0.0.127.in-addr.arpa    /etc/named.local
```

Caching Only Server.

```
;
; Boot file for Caching Master 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
```

B.5.7 Remote Server

/etc/resolv.conf

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

B.5.8 named.ca

```
;
; Initial cache data for root domain servers.
;

.                    99999999   IN   NS   USC-ISIC.ARPA.
                     99999999   IN   NS   USC-ISIB.ARPA.
                     99999999   IN   NS   BRL-AOS.ARPA.
                     99999999   IN   NS   SRI-NIC.ARPA.
;  Prep the cache (hotwire the addresses).
SRI-NIC.ARPA.        99999999   IN   A    10.0.0.51
USC-ISIB.ARPA.       99999999   IN   A    10.3.0.52
USC-ISIC.ARPA.       99999999   IN   A    10.0.0.52
BRL-AOS.ARPA.        99999999   IN   A    128.20.1.2
BRL-AOS.ARPA.        99999999   IN   A    192.5.22.82
```

B.5.9 named.local

```
@    IN    SOA    ucbvax.Berkeley.Edu. kjd.ucbvax.Berkeley.Edu. (
                  1       ; Serial
                  3600    ; Refresh
                  300     ; Retry
                  3600000 ; Expire
                  3600 )  ; Minimum
     IN    NS     ucbvax.Berkeley.Edu.
1    IN    PTR    localhost.
```

B.5.10 UCB 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        127.1
ucbarpa         IN      A        128.32.4
                IN      A        10.0.0.78
                ANY     HINFO    VAX-11/780 UNIX
arpa            IN      CNAME    ucbarpa
ernie           IN      A        128.32.6
                ANY     HINFO    VAX-11/780 UNIX
ucbernie        IN      CNAME    ernie
monet           IN      A        128.32.7
                IN      A        128.32.130.6
                ANY     HINFO    VAX-11/750 UNIX
ucbmonet        IN      CNAME    monet
ucbvax          IN      A        10.2.0.78
                IN      A        128.32.10
                ANY     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
                ANY     HINFO   Pro350 RT11
toybox          IN      MX      0  monet.Berkeley.Edu
miriam          ANY     MB      vineyd.DEC.COM.
postmistress    ANY     MR      Miriam
Bind            ANY     MINFO   Bind-Request kjd.Berkeley.Edu.
                ANY     MG      Ralph.Berkeley.Edu.
                ANY     MG      Zhou.Berkeley.Edu.
                ANY     MG      Painter.Berkeley.Edu.
                ANY     MG      Riggle.Berkeley.Edu.
                ANY     MG      Terry.pa.Xerox.Com.
```

B.5.11 UCB Host.rev

```
;
;  @(#)ucb-hosts.rev   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.
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.
```

B.6 Domain Management

This section contains information for starting, controlling and debugging **named**.

B.6.1 /etc/rc.local

The hostname should be set to the full domain style name in **/etc/rc.local** using **hostname**. The following entry should be added to */etc/rc.local* to start up **named** at system boot time:

```
if [ -f /etc/named ]; then
    /etc/named [options] & echo -n ' named' >/dev/console
fi
```

This usually directly follows the lines that start **syslogd**. **Do Not** attempt to run **named** from **inetd**. This will continuously restart the name server and defeat the purpose of having a cache.

B.6.2 /etc/named.pid

When **named** is successfully started up it writes its process id into the file */etc/named.pid*. This is useful to programs that want to send signals to **named**. The name of this file may be changed by defining PIDFILE to the new name when compiling **named**.

B.6.3 /etc/hosts

The **gethostbyname** library call can detect if **named** is running. If it is determined that **named** is not running it will look in */etc/hosts* to resolve an address. This option was added to allow **/etc/init.d/tcp** to configure the machines 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 machines interface addresses and a couple of machine names and address in */etc/hosts* so the system manager can rcp files from another machine when the system is in single user mode. The format of */etc/hosts* has not changed. See the **hosts** manual page for more information. Since the process of reading */etc/hosts* is slow, it is not advised to use this option when the system is in multi user mode.

B.6.4 Signals

There are several signals that can be sent to the **named** process to have it do tasks without restarting the process.

Reload

SIGHUP - 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/messages* and */usr/spool/adm/syslog* to check for any messages logged by *syslog*. Next send it a signal to see what is happening.

SIGINT - Dumps the current data base and cache to */usr/tmp/named_dump.db*. This should give you an indication to whether the data base was loaded correctly. The name of the dump file may be changed by defining DUMPFILE to the new name when compiling **named**.

Note: the following two signals only work when **named** is built with DEBUG defined.

SIGUSR1 - Turns on debugging. Each following USR1 increments the debug level. The output goes to */usr/tmp/named.run* The name of this debug file may be changed by defining DEBUGFILE to the new name before compiling **named**.

SIGUSR2 - Turns off debugging completely.

For more detailed debugging, define DEBUG when compiling the resolver routines into */lib/libc.a*.

ACKNOWLEDGEMENTS

Many thanks to the users at U.C. Berkeley for falling into many of the holes involved with integrating BIND into the system so that others would be spared the trauma. I would also like to extend gratitude to Jim McGinness and Digital Equipment Corporation for permitting me to spend most of my time on this project.

Ralph Campbell, Doug Kingston, Craig Partridge, Smoot Carl-Mitchell, Mike Muuss and everyone else on the DARPA Internet who has contributed to the development of BIND. To the members of the original BIND project, Douglas Terry, Mark Painter, David Riggle and Songnian Zhou.

Anne Hughes, Jim Bloom and Kirk McKusick and the many others who have reviewed this paper giving considerable advice.

This work was sponsored by the Defense Advanced Research Projects Agency (DoD), Arpa Order No. 4871 monitored by the Naval Electronics Systems Command under contract No. N00039-84-C-0089. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Research Projects Agency, of the US Government, or of Digital Equipment Corporation.

C. Sendmail Overview

C. Sendmail Overview

Appendix C
Sendmail Overview

C.1 **Abstract** C-1

C.2 **Introduction** C-1

C.3 **Design Goals** C-2

C.4 **Overview** C-4
- System Organization C-4
- Interfaces to the Outside World C-4
  - Argument vector/exit status C-5
  - SMTP over pipes C-5
  - SMTP over an IPC connection C-5
- Operational Description C-5
  - Argument processing and address parsing C-5
  - Message collection C-6
  - Message delivery C-6
  - Queueing for retransmission C-6
  - Return to sender C-6
- Message Header Editing C-7
- Configuration File C-7

C.5 **Usage and Implementation** C-7
- Arguments C-7
- Mail to Files and Programs C-8
- Aliasing, Forwarding, Inclusion C-8
  - Aliasing C-8
  - Forwarding C-8
  - Inclusion C-9
- Message Collection C-9
- Message Delivery C-10
- Queued Messages C-10
- Configuration C-10
  - Macros C-11
  - Header declarations C-11
  - Mailer declarations C-11
  - Address rewriting rules C-11
  - Option setting C-12

C.6 Comparison With Other Mailers C-12
Delivermail C-12
MMDF C-13
Message Processing Module C-13

C.7 Evaluations and Future Plans C-14

Appendix C
Sendmail Overview

This appendix provides a detailed overview of the Sendmail. It is taken from *SENDMAIL – An Internetwork Mail Router* by Eric Allman.

C.1 Abstract

Routing mail through a heterogeneous internet presents many new problems. Among the worst of these is that of address mapping. Historically, this has been handled on an *ad hoc* basis. However, this approach has become unmanageable as internets grow.

Sendmail acts a unified “post office” to which all mail can be submitted. Address interpretation is controlled by a production system, which can parse both domain-based addressing and old-style *ad hoc* addresses. The production system is powerful enough to rewrite addresses in the message header to conform to the standards of a number of common target networks, including old (NCP/RFC733) Arpanet, new (TCP/RFC822) Arpanet, UUCP, and Phonenet. Sendmail also implements an SMTP server, message queueing, and aliasing.

C.2 Introduction

Sendmail implements a general internetwork mail routing facility, featuring aliasing and forwarding, automatic routing to network gateways, and flexible configuration.

In a simple network, each node has an address, and resources can be identified with a host-resource pair; in particular, the mail system can refer to users using a host-username pair. Host names and numbers have to be administered by a central authority, but usernames can be assigned locally to each host.

In an internet, multiple networks with different characteristics and managements must communicate. In particular, the syntax and semantics of resource identification change. Certain special cases can be handled trivially by *ad hoc* techniques, such as providing network names that appear local to hosts on other networks, as with the Ethernet at Xerox PARC. However, the general case is extremely complex. For example, some networks require point-to-point routing, which simplifies the database update problem since only adjacent hosts must be entered into the system tables, while others use end-to-end addressing. Some networks use a left-associative syntax and others use a right-associative syntax, causing ambiguity in mixed addresses.

Internet standards seek to eliminate these problems. Initially, these proposed expanding the address pairs to address triples, consisting of {network, host, resource} triples. Network numbers must be universally agreed upon, and hosts can be assigned locally on each network. The user-level presentation was quickly expanded to address domains, comprised of a local resource identification and a hierarchical domain specification with a common static root. The domain technique separates the issue of physical versus logical addressing. For example, an address of the form `eric@a.cc.berkeley.arpa` describes only the logical organization of the address space.

Sendmail is intended to help bridge the gap between the totally *ad hoc* world of networks that know nothing of each other and the clean, tightly-coupled world of unique network numbers. It can accept old arbitrary address syntaxes, resolving ambiguities using heuristics specified by the system administrator, as well as domain-based addressing. It helps guide the conversion of message formats between disparate networks. In short, Sendmail is designed to assist a graceful transition to consistent internetwork addressing schemes.

Section C.3 discusses the design goals for Sendmail. Section C.4 gives an overview of the basic functions of the system. In section C.5, details of usage are discussed. Section C.6 compares Sendmail to other internet mail routers, and an evaluation of Sendmail is given in section C.7, including future plans.

C.3 Design Goals

Design goals for Sendmail include:

- Compatibility with the existing mail programs, including Bell version 6 mail, Bell version 7 mail [UNIX83], Berkeley Mail [Shoens79], BerkNet mail [Schmidt79], and hopefully UUCP mail [Nowitz78a, Nowitz78b]. ARPANET mail [Crocker77a, Postel77] was also required.
- Reliability, in the sense of guaranteeing that every message is correctly delivered or at least brought to the attention of a human for correct disposal; no message should ever be completely lost. This goal was considered essential because of the emphasis on mail in our environment. It has turned out to be one of the hardest goals to satisfy, especially in the face of the many anomalous message formats produced by various ARPANET sites. For example, certain sites generate improperly formated addresses, occasionally causing error-message loops. Some hosts use blanks in names, causing problems with UNIX mail programs that as-

sume that an address is one word. The semantics of some fields are interpreted slightly differently by different sites. In summary, the obscure features of the ARPANET mail protocol really are used and are difficult to support, but must be supported.

- Existing software to do actual delivery should be used whenever possible. This goal derives as much from political and practical considerations as technical.
- Easy expansion to fairly complex environments, including multiple connections to a single network type (such as with multiple UUCP or Ether nets [Metcalfe76]). This goal requires consideration of the contents of an address as well as its syntax in order to determine which gateway to use. For example, the ARPANET is bringing up the TCP protocol to replace the old NCP protocol. No host at Berkeley runs both TCP and NCP, so it is necessary to look at the ARPANET host name to determine whether to route mail to an NCP gateway or a TCP gateway.
- Configuration should not be compiled into the code. A single compiled program should be able to run as is at any site (barring such basic changes as the CPU type or the operating system). We have found this seemingly unimportant goal to be critical in real life. Besides the simple problems that occur when any program gets recompiled in a different environment, many sites like to fiddle with anything that they will be recompiling anyway.
- Sendmail must be able to let various groups maintain their own mailing lists, and let individuals specify their own forwarding, without modifying the system alias file.
- Each user should be able to specify which mailer to execute to process mail being delivered for him. This feature allows users who are using specialized mailers that use a different format to build their environment without changing the system, and facilitates specialized functions (such as returning an "I am on vacation" message).
- Network traffic should be minimized by batching addresses to a single host where possible, without assistance from the user.

These goals motivated the architecture illustrated in Figure 1. The user interacts with a mail generating and sending program. When the mail is created, the generator calls Sendmail, which routes the message to the correct mailer(s). Since some of the senders may be network servers and some of the mailers may be network clients, Sendmail may be used as an internet mail gateway.

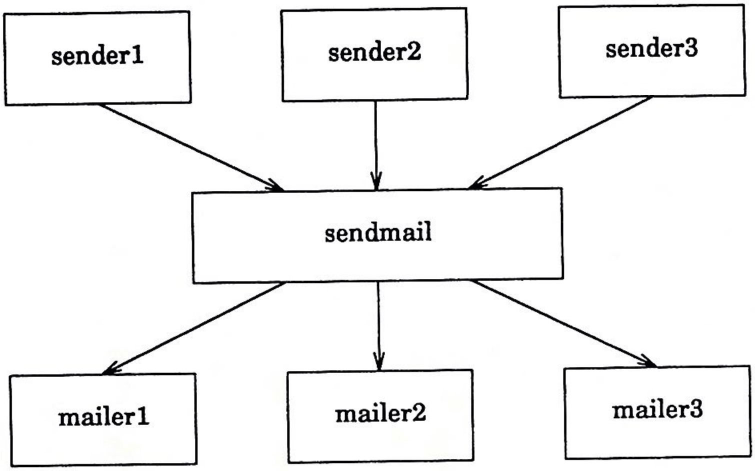


Figure 1 Sendmail system structure.

C.4 Overview

C.4.1 System Organization

Sendmail neither interfaces with the user nor does actual mail delivery. Rather, it collects a message generated by a user interface program (UIP) such as Berkeley *Mail*, MS [Crocker77b], or MH [Borden79], edits the message as required by the destination network, and calls appropriate mailers to do mail delivery or queueing for network transmission[1]. This discipline allows the insertion of new mailers at minimum cost. In this sense Sendmail resembles the Message Processing Module (MPM) of [Postel79b].

C.4.2 Interfaces to the Outside World

There are three ways Sendmail can communicate with the outside world, both in receiving and in sending mail. These are using the conventional UNIX argument vector/return status, speaking SMTP over a pair of UNIX pipes, and speaking SMTP over an interprocess(or) channel.

---

[1] except when mailing to a file, when Sendmail does the delivery directly.

Argument vector/exit status

This technique is the standard UNIX method for communicating with the process. A list of recipients is sent in the argument vector, and the message body is sent on the standard input. Anything that the mailer prints is simply collected and sent back to the sender if there were any problems. The exit status from the mailer is collected after the message is sent, and a diagnostic is printed if appropriate.

SMTP over pipes

The SMTP protocol [Postel82] can be used to run an interactive lock-step interface with the mailer. A subprocess is still created, but no recipient addresses are passed to the mailer via the argument list. Instead, they are passed one at a time in commands sent to the processes standard input. Anything appearing on the standard output must be a reply code in a special format.

SMTP over an IPC connection

This technique is similar to the previous technique, except that it uses a 4.2bsd IPC channel [UNIX83]. This method is exceptionally flexible in that the mailer need not reside on the same machine. It is normally used to connect to a sendmail process on another machine.

C.4.3 Operational Description

When a sender wants to send a message, it issues a request to Sendmail using one of the three methods described above. Sendmail operates in two distinct phases. In the first phase, it collects and stores the message. In the second phase, message delivery occurs. If there were errors during processing during the second phase, Sendmail creates and returns a new message describing the error and/or returns an status code telling what went wrong.

Argument processing and address parsing

If Sendmail is called using one of the two subprocess techniques, the arguments are first scanned and option specifications are processed. Recipient addresses are then collected, either from the command line or from the SMTP RCPT command, and a list of recipients is created. Aliases are expanded at this step, including mailing lists. As much validation as possible of the addresses is done at this step: 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.

Sendmail appends each address to the recipient list after parsing. 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 delivered to the same recipient, as might occur if a person is in two groups.

Message collection

Sendmail then collects the message. 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 (i.e., binary data is not allowed). 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 were valid. The message will be returned with an error.

Message delivery

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 only accept one recipient at a time are handled properly.

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 prepended by a customized header. The mailer status code is caught and checked, and a suitable error message given as appropriate. The exit code must conform to a system standard or a generic message ("Service unavailable") is given.

Queueing for retransmission

If the mailer returned a status that indicated that it might be able to handle the mail later, Sendmail will queue the mail and try again later.

Return to sender

If errors occur during processing, Sendmail returns the message to the sender for retransmission. The letter can be mailed back or written in the file *dead.letter* in the sender's home directory[2].

---

[2] Obviously, if the site giving the error is not the originating site, the only reasonable option is to mail back to the sender. Also, there are many more error disposition options, but they only effect the error message – the "return to sender" function is always handled in one of these two ways.

C.4.4 Message Header Editing

Certain editing of the message header occurs automatically. 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.

C.4.5 Configuration File

Almost all configuration information is read at runtime from an ASCII file, encoding macro definitions (defining the value of macros used internally), header declarations (telling sendmail the format of header lines that it will process specially, i.e., lines that it will add or reformat), mailer definitions (giving information such as the location and characteristics of each mailer), and address rewriting rules (a limited production system to rewrite addresses which is used to parse and rewrite the addresses).

To improve performance when reading the configuration file, a memory image can be provided. This provides a compiled form of the configuration file.

C.5 Usage and Implementation

C.5.1 Arguments

Arguments may be flags and addresses. Flags set various processing options. Following flag arguments, address arguments may be given, unless we are running in SMTP mode. Addresses follow the syntax in RFC822 [Crocker82] for ARPANET address formats. In brief, the format is:

- Anything in parentheses is thrown away (as a comment).
- Anything in angle brackets ("< >") is preferred over anything else. This rule implements the ARPANET standard that addresses of the form

  ```
  user name <machine-address>
  ```

  will send to the electronic machine-address rather than the human "user name."
- Double quotes (") quote phrases; backslashes quote characters. Backslashes are more powerful in that they will cause otherwise equivalent phrases to compare differently – for example, *user* and *"user"* are equivalent, but *user* is different from either of them.

Parentheses, angle brackets, and double quotes must be properly balanced and nested. The rewriting rules control remaining parsing[3].

C.5.2 Mail to Files and Programs

Files and programs are legitimate message recipients. Files provide archival storage of messages, useful for project administration and history. Programs are useful as recipients in a variety of situations, for example, to maintain a public repository of systems messages (such as the Berkeley **msgs** program, or the MARS system [Sattley78]).

Any address passing through the initial parsing algorithm as a local address (i.e, not appearing to be a valid address for another mailer) is scanned for two special cases. If prefixed by a vertical bar (|) the rest of the address is processed as a shell command. If the user name begins with a slash mark (/) the name is used as a file name, instead of a login name.

Files that have setuid or setgid bits set but no execute bits set have those bits honored if Sendmail is running as root.

C.5.3 Aliasing, Forwarding, Inclusion

Sendmail reroutes mail three ways. Aliasing applies system wide. Forwarding allows each user to reroute incoming mail destined for that account. Inclusion directs Sendmail to read a file for a list of addresses, and is normally used in conjunction with aliasing.

Aliasing

Aliasing maps 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 (since there are no nicknames for the local host).

Forwarding

After aliasing, recipients that are local and valid are checked for the existence of a file in their home directory. If it exists, the message is not sent to that user, but rather to the list of users in that file. Often this list will contain only one address, and the feature will be used for network mail forwarding.

---

[3] Disclaimer: Some special processing is done after rewriting local names; see below.

Forwarding also permits a user to specify a private incoming mailer. For example, forwarding to:

```
|/usr/local/newmail myname
```

will use a different incoming mailer.

Inclusion

Inclusion is specified in RFC 733 [Crocker77a] syntax:

```
:Include: pathname
```

An address of this form reads the file specified by *pathname* and sends to all users listed in that file.

The intent is not to support direct use of this feature, but rather to use this as a subset of aliasing. For example, an alias of the form:

```
project: :include:/usr/project/userlist
```

is a method of letting a project maintain a mailing list without interaction with the system administration, even if the alias file is protected.

It is not necessary to rebuild the index on the alias database when a :include: list is changed.

C.5.4 Message Collection

Once all recipient addresses are parsed and verified, the message is collected. The message comes in two parts: a message header and a message body, separated by a blank line. 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 following lines with a space or a tab. Some header fields have special internal meaning, and have appropriate special processing. Other headers are simply passed through. Some header fields may be added automatically, such as time stamps.

The body is a series of text lines. It is completely uninterpreted and untouched, 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.

C.5.5 Message Delivery

The send queue is ordered by receiving host before transmission to implement message batching. Each address is marked as it is sent 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. The interface to the mailer is performed using one of the techniques described in section C.4.2.

After a connection is established, Sendmail makes the per-mailer changes to the header and sends the result to the mailer. If any mail is rejected by the mailer, a flag is set to invoke the return-to-sender function after all delivery completes.

C.5.6 Queued Messages

If the mailer returns a "temporary failure" exit status, the message is queued. A control file is used to describe 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 salient 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.

C.5.7 Configuration

Configuration is controlled primarily by a configuration file read at startup. Sendmail should not need to be recomplied except:

- To change operating systems (V6, V7/32V, 4BSD).
- To remove or insert the DBM (UNIX database) library.
- To change ARPANET reply codes.
- To add headers fields requiring special processing.

Adding mailers or changing parsing (i.e., rewriting) or routing information does not require recompilation.

If the mail is being sent by a local user, and the file exists in the sender's home directory, that file is read as a configuration file after the system configuration file. The primary use of this feature is to add header lines.

The configuration file encodes macro definitions, header definitions, mailer definitions, rewriting rules, and options.

Macros

Macros can be used in three ways. Certain macros transmit unstructured textual information into the mail system, such as the name Sendmail will use to identify itself in error messages. Other macros transmit information from Sendmail to the configuration file for use in creating other fields (such as argument vectors to mailers); e.g., the name of the sender, and the host and user of the recipient. Other macros are unused internally, and can be used as shorthand in the configuration file.

Header declarations

Header declarations inform Sendmail of the format of known header lines. Knowledge of a few header lines is built into Sendmail, such as the `From:` and `Date:` lines.

Most configured headers will be automatically inserted in the outgoing message if they don't exist in the incoming message. Certain headers are suppressed by some mailers.

Mailer declarations

Mailer declarations tell Sendmail of the various mailers available to it. The definition specifies the internal name of the mailer, the pathname of the program to call, some flags associated with the mailer, and an argument vector to be used on the call; this vector is macro-expanded before use.

Address rewriting rules

The heart of address parsing in Sendmail is a set of rewriting rules. These are an ordered list of pattern-replacement rules, (somewhat like a production system, except that order is critical), which are applied to each address. The address is rewritten textually until it is either rewritten into a special canonical form (i.e., a (mailer, host, user) 3-tuple, such as {arpanet, usc-isif, postel} representing the address "postel@usc-isif"), or it falls off the end. When a pattern matches, the rule is reapplied until it fails.

The configuration file also supports the editing of addresses into different formats. For example, an address of the form:

```
ucsfcgl!tef
```

might be mapped into:

```
tef@ucsfcgl.UUCP
```

to conform to the domain syntax. Translations can also be done in the other direction.

Option setting

There are several options that can be set from the configuration file. These include the pathnames of various support files, timeouts, default modes, etc.

C.6 Comparison With Other Mailers

C.6.1 Delivermail

Sendmail is an outgrowth of *delivermail*. The primary differences are:

- Configuration information is not compiled in. This change simplifies many of the problems of moving to other machines. It also allows easy 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.
- Forwarding and `:include:` features 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).
- Sendmail 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.
- Sendmail uses the networking support provided by 4.2BSD to provide a direct interface networks such as the ARPANET and/or Ethernet using SMTP (the Simple Mail Transfer Protocol) over a TCP/IP connection.

C.6.2 MMDF

MMDF [Crocker79] spans a wider problem set than Sendmail. For example, the domain of MMDF includes a "phone network" mailer, whereas Sendmail calls on preexisting 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.

The configuration for MMDF is compiled into the code[4].

Since 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[5] into MMDF. In particular, MMDF must know the location and format of host tables for all channels, and the 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.

C.6.3 Message Processing Module

The Message Processing Module (MPM) discussed by Postel [Postel79b] 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, thus allowing 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 tuples[6]. 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.

---

[4] Dynamic configuration tables are currently being considered for MMDF; allowing the installer to select either compiled or dynamic tables.

[5] The MMDF equivalent of a Sendmail mailer.

[6] This is similar to the NBS standard.

C.7 Evaluations and Future Plans

Sendmail is designed to work in a nonhomogeneous environment. Every attempt is made to avoid imposing unnecessary constraints on the underlying mailers. This goal has driven much of the design. One of the major problems has been the lack of a uniform address space, as postulated in [Postel79a] and [Postel79b].

A nonuniform address space implies that a path will be specified in all addresses, either explicitly (as part of the address) or implicitly (as with implied forwarding to gateways). This restriction has the unpleasant effect of making replying to messages exceedingly difficult, since there is no one address for any person, but only a way to get there from wherever you are.

Interfacing to mail programs that were not initially intended to be applied in an internet environment has been amazingly successful, and has reduced the job to a manageable task.

Sendmail has knowledge of a few difficult environments built in. It generates ARPANET FTP/SMTP compatible error messages (prepended with three-digit numbers [Neigus73, Postel74, Postel82]) as necessary, optionally generates UNIX-style `From` lines on the front of messages for some mailers, and knows how to parse the same lines on input. Also, error handling has an option customized for BerkNet.

The decision to avoid doing any type of delivery where possible (even, or perhaps especially, local delivery) has turned out to be a good idea. Even with local delivery, there are issues of the location of the mailbox, the format of the mailbox, the locking protocol used, etc., that are best decided by other programs. One surprisingly major annoyance in many internet mailers is that the location and format of local mail is built in. The feeling seems to be that local mail is so common that it should be efficient. This feeling is not born out by our experience; on the contrary, the location and format of mailboxes seems to vary widely from system to system.

The ability to automatically generate a response to incoming mail (by forwarding mail to a program) seems useful ("I am on vacation until late August....") but can create problems such as forwarding loops (two people on vacation whose programs send notes back and forth, for instance) if these programs are not well written. A program could be written to do standard tasks correctly, but this would solve the general case.

It might be desirable to implement some form of load limiting. I am unaware of any mail system that addresses this problem, nor am I aware of any reasonable solution at this time.

The configuration file is currently practically inscrutable; considerable convenience could be realized with a higher-level format.

It seems clear that common protocols will be changing soon to accommodate changing requirements and environments. These changes will include modifications to the message header (e.g., [NBS80]) or to the body of the message itself (such as for multimedia messages [Postel80]). Experience indicates that these changes should be relatively trivial to integrate into the existing system.

In tightly coupled environments, it would be nice to have a name server such as Grapvine [Birrell82] integrated into the mail system. This would allow a site such as Berkeley to appear as a single host, rather than as a collection of hosts, and would allow people to move transparently among machines without having to change their addresses. Such a facility would require an automatically updated database and some method of resolving conflicts. Ideally this would be effective even without all hosts being under a single management. However, it is not clear whether this feature should be integrated into the aliasing facility or should be considered a "value added" feature outside Sendmail itself.

As a more interesting case, the CSNET name server [Solomon81] provides an facility that goes beyond a single tightly-coupled environment. Such a facility would normally exist outside of Sendmail however.

ACKNOWLEDGEMENTS

Thanks are due to Kurt Shoens for his continual cheerful assistance and good advice, Bill Joy for pointing me in the correct direction (over and over), and Mark Horton for more advice, prodding, and many of the good ideas. Kurt and Eric Schmidt are to be credited for using *delivermail* as a server for their programs (*Mail* and BerkNet respectively) before any sane person should have, and making the necessary modifications promptly and happily. Eric gave me considerable advice about the perils of network software which saved me an unknown amount of work and grief. Mark did the original implementation of the DBM version of aliasing, installed the VFORK code, wrote the current version of *rmail*, and was the person who really convinced me to put the work into *delivermail* to turn it into Sendmail. Kurt deserves accolades for using Sendmail when I was myself afraid to take the risk; how a person can continue to be so enthusiastic in the face of so much bitter reality is beyond me.

Kurt, Mark, Kirk McKusick, Marvin Solomon, and many others have reviewed this paper, giving considerable useful advice.

Special thanks are reserved for Mike Stonebraker at Berkeley and Bob Epstein at Britton-Lee, who both knowingly allowed me to put so much work into this project when there were so many other things I really should have been working on.

D. Sendmail Installation
Operation Guide

D. *Sendmail Installation*
Operation Guide

Appendix D

Sendmail Installation and Operation Guide

D.1 Introduction D-1

D.2 Basic Installation D-2
- Off-The-Shelf Configurations D-2
- Installation Using the Makefile D-3
- Installation by Hand D-3
  - /usr/lib/sendmail D-3
  - /usr/lib/sendmail.cf D-4
  - /usr/lib/newaliases D-4
  - /usr/spool/mqueue D-4
  - /usr/lib/aliases* D-4
  - /usr/lib/sendmail.fc D-5
  - /etc/rc D-5
  - /usr/lib/sendmail.hf D-5
  - /usr/lib/sendmail.st D-6
  - /usr/lib/newaliases D-6
  - /usr/lib/mailq D-6

D.3 Normal Operations D-6
- Quick Configuration Startup D-6
- The System Log D-7
  - Format D-7
  - Levels D-7
- The Mail Queue D-7
  - Printing the queue D-7
  - Format of queue files D-7
  - Forcing the queue D-9
- The Alias Database D-10
  - Rebuilding the alias database D-10
  - Potential problems D-11
  - List owners D-11
- Per-User Forwarding (.forward Files) D-12
- Special Header Lines D-12
  - Return-Receipt-To: D-12
  - Errors-To: D-12
  - Apparently-To: D-13

Sending 8-bit mail D-13

D.4 Arguments D-13
Queue Interval D-13
Daemon Mode D-14
Forcing the Queue D-14
Debugging D-14
Trying a Different Configuration File D-15
Changing the Values of Options D-15

D.5 Sendmail Tuning D-15
Timeouts D-15
  Queue interval D-16
  Read timeouts D-16
  Message timeouts D-16
Forking During Queue Runs D-16
Queue Priorities D-17
Load Limiting D-17
Delivery Mode D-18
Log Level D-18
File Modes D-19
  To suid or not suid D-19
  Should my alias database be writable? D-19

D.6 The Whole Scoop on The Configuration File D-20
The Syntax D-20
  R and S Rewriting Rules D-20
  D— define macro D-21
  C and F— define classes D-21
  M— define mailer D-21
  H— define header D-22
  O— set option D-22
  T— define trusted users D-23
  P — precedence definitions D-23
The Semantics D-23
  Special Macros and Conditionals D-23
  Special Classes D-26
  The left hand side D-26
  The right hand side D-26
  Semantics of Rewriting Rule Sets D-28
  Mailer flags D-29
  The error mailer D-29
Building a Configuration File From Scratch D-29
  What you are trying to do D-29

Philosophy D-30
Relevant issues D-31
How to proceed D-32
Testing the rewriting rules — the –bt flag D-32
Building mailer descriptions D-33

D.7 Command Line Flags D-35

D.8 Configuration Options D-36

D.9 Mailer Flags D-40

D.10 Other Configuration D-41
Parameters in src/conf.h D-42
Configuration in src/conf.c D-44
Configuration in src/daemon.c D-47

D.11 Summary of Support Files D-47

Appendix D
Sendmail Installation and Operation Guide

This appendix provides a detailed description of Sendmail configuration. It also provides information on the configuration file. The information in this appendix is taken from the *SENDMAIL Installation and Operation Guide* by Eric Allman.

D.1 Introduction

Sendmail implements a general purpose internetwork mail routing facility under the UNIX* operating system. It is not tied to any one transport protocol – its function may be likened to a crossbar switch, relaying messages from one domain into another. In the process, it can do a limited amount of message header editing to put the message into a format that is appropriate for the receiving domain. All of this is done under the control of a configuration file.

Due to the requirements of flexibility for Sendmail, the configuration file can seem somewhat unapproachable. However, there are only a few basic configurations for most sites, for which standard configuration files have been supplied. Most other configurations can be built by adjusting an existing configuration files incrementally.

Although Sendmail is intended to run without the need for monitoring, it has a number of features that may be used to monitor or adjust the operation under unusual circumstances. These features are described.

Section D.2 describes how to do a basic Sendmail installation. Section D.3 explains the day-to-day information you should know to maintain your mail system. If you have a relatively normal site, these two sections should contain sufficient information for you to install Sendmail and keep it happy. Section D.4 describes some parameters that may be safely adjusted. Section D.5 has information regarding the command line arguments. Section D.6 contains the information about the configuration file. This section is for people who must write their own configuration file. Sections D.7, D.8, D.9, D.10, and D.11 give a brief but detailed explanation of a number of features not described in the rest of the paper.

---

* UNIX is a trademark of Bell Laboratories.

The references in this paper are actually found in the companion paper *Sendmail – An Internetwork Mail Router*. This other paper should be read before this manual to gain a basic understanding of how the pieces fit together.

D.2 Basic Installation

There are two basic steps to installing Sendmail. The hard part is to build the configuration table. This is a file that sendmail reads when it starts up that describes the mailers it knows about, how to parse addresses, how to rewrite the message header, and the settings of various options. Although the configuration table is quite complex, a configuration can usually be built by adjusting an existing off-the-shelf configuration. The second part is actually doing the installation, i.e., creating the necessary files, etc. The remainder of this section will describe the installation of sendmail assuming you can use one of the existing configurations and that the standard installation parameters are acceptable. All pathnames and examples are given from the root of the Sendmail subtree, normally */usr/src/usr.lib/sendmail* on 4.3BSD.

D.2.1 Off-The-Shelf Configurations

Configuration files currently in use at Berkeley are in the directory *cf* of the sendmail directory. This directory contains three subdirectories: *cf*, *m4*, and *sitedep*. The directory *cf/m4* contains site-independent *m4* include files that have information common to all configuration files, while *cf/sitedep* contains *m4* include files that have site-specific information in them. These files are used by the master configuration (*.mc*) in *cf/cf* and produce standard configuration files (with ".cf" suffix) when run through *m4*.

Three off-the-shelf configurations are supplied to handle the basic cases:

- Internet sites running the nameserver (or using host tables wherein the fully-qualified domain name of each host is listed first) can use *cf/tcpproto.cf*. For simple sites, you should be able to use this file without modification. This file is not in *m4* format.
- UUCP only sites can use *cf/uucpproto.cf*. This file is not in *m4* format.
- A group of machines at a single site connected by an Ethernet (or other networking that supports TCP/IP) with (only) one host connected to the outside world via UUCP is represented by two configuration files: *tcpuucpproto.cf* should be installed on the host with outside connections and *tcpproto.cf* should be installed on all other hosts.

Some configuration will be needed in each of the above cases. Just be sure to correctly fill in the blanks as shown in the instructions in the configuration file. Then install the file as */usr/lib/sendmail.cf*.

If you are running a larger or more complex site, it is to your advantage to read the *README* file in the *cf* subdirectory. This file explains how to use *m4* to automatically create configuration files for non-standard situations.

D.2.2 Installation Using the Makefile

A makefile exists in the root of the Sendmail directory that will do all of these steps for a 4.3BSD system. It may have to be slightly tailored for use on other systems.

Before using this makefile, you should create a symbolic link from *cf* to the directory containing your configuration files. You should also have created your configuration file and left it in the file *cf/system.cf* where `system` is the name of your system (i.e., what is returned by **hostname** If you do not have **hostname** you can use the declaration `HOST=`*system* on the **make** command line. You should also examine the file *md/config.m4* and change the **m4** macros there to reflect any libraries and compilation flags you may need.

The basic installation procedure is to type:

```
make
make install
make installcf
```

in the root directory of the Sendmail distribution. This will make all binaries and install them in the standard places. The second and third **make** commands must be executed as the superuser (root).

D.2.3 Installation by Hand

Along with building a configuration file, you will have to install the Sendmail startup into your UNIX system. If you are doing this installation in conjunction with a regular Berkeley UNIX install, these steps will already be complete. Many of these steps will have to be executed as the superuser (root).

/usr/lib/sendmail

The binary for sendmail is located in */usr/lib*. If it becomes necessary to recompile and reinstall the entire system, the following sequence will do it:

```
cd src
make clean
make install
```

/usr/lib/sendmail.cf

The configuration file that you created earlier should be installed in */usr/lib/sendmail.cf*:

```
cp cf/system.cf /usr/lib/sendmail.cf
```

/usr/lib/newaliases

If you are running delivermail, it is critical that the **newaliases** command be replaced. This can just be a link to Sendmail

```
rm /usr/lib/newaliases
ln /usr/lib/sendmail /usr/lib/newaliases
```

/usr/spool/mqueue

The directory */usr/spool/mqueue* should be created to hold the mail queue. This directory should be mode 755 and owned by root.

/usr/lib/aliases*

The system aliases are held in three files. The file */usr/lib/aliases* is the master copy. A sample is given in *lib/aliases* which includes some aliases which *must* be defined:

```
cp lib/aliases /usr/lib/aliases
```

You should extend this file with any aliases that are apropos to your system.

Normally Sendmail looks at a version of these files maintained by the **dbm** routines. These are stored in */usr/lib/aliases.dir* and */usr/lib/aliases.pag*. These can initially be created as empty files, but they will have to be initialized promptly. These should be mode 644:

```
cp /dev/null /usr/lib/aliases.dir
cp /dev/null /usr/lib/aliases.pag
chmod 644 /usr/lib/aliases.*
newaliases
```

/usr/lib/sendmail.fc

If you intend to install the frozen version of the configuration file (for quick startup) you should create the file */usr/lib/sendmail.fc* and initialize it. This step may be safely skipped.

```
cp /dev/null /usr/lib/sendmail.fc
/usr/lib/sendmail -bz
```

/etc/rc

It will be necessary to start up the sendmail daemon when your system reboots. This daemon performs two functions: it listens on the SMTP socket for connections (to receive mail from a remote system) and it processes the queue periodically to insure that mail gets delivered when hosts come up.

Add the following lines to */etc/rc* (or */etc/rc.local* as appropriate) in the area where it is starting up the daemons:

```
if [ -f /usr/lib/sendmail ]; then
        (cd /usr/spool/mqueue; rm -f [lnx]f*)
        /usr/lib/sendmail -bd -q30m &
        echo -n ' sendmail' >/dev/console
```

The **cd** and **rm** commands insure that all lock files have been removed; extraneous lock files may be left around if the system goes down in the middle of processing a message. The line that actually invokes Sendmail has two flags: **–bd** causes it to listen on the SMTP port, and **–q30m** causes it to run the queue every half hour.

If you are not running a version of UNIX that supports Berkeley TCP/IP, do not include the **–bd** flag.

/usr/lib/sendmail.hf

This is the help file used by the SMTP **HELP** command. It should be copied from *lib/sendmail.hf*:

```
cp lib/sendmail.hf /usr/lib
```

/usr/lib/sendmail.st

If you wish to collect statistics about your mail traffic, you should create the file */usr/lib/sendmail.st*:

```
cp /dev/null /usr/lib/sendmail.st
chmod 666 /usr/lib/sendmail.st
```

This file does not grow. It is printed with the program *aux/mailstats*.

/usr/lib/newaliases

If Sendmail is invoked as **newaliases**, it will simulate the **–bi** flag (i.e., will rebuild the alias database; see below). This should be a link to */usr/lib/sendmail.*

/usr/lib/mailq

If Sendmail is invoked as **mailq**, it will simulate the **–bp** flag (i.e., Sendmail will print the contents of the mail queue; see below). This should be a link to */usr/lib/sendmail.*

D.3 Normal Operations

D.3.1 Quick Configuration Startup

A fast version of the configuration file may be set up by using the **–bz** flag:

```
/usr/lib/sendmail -bz
```

This creates the file */usr/lib/sendmail.fc* ("frozen configuration"). This file is an image of Sendmail's data space after reading in the configuration file. If this file exists, it is used instead of */usr/lib/sendmail.cf*. *sendmail.fc* must be rebuilt manually every time *sendmail.cf* is changed.

The frozen configuration file will be ignored if a **–C** flag is specified or if Sendmail detects that it is out of date. However, the heuristics are not strong so this should not be trusted.

D.3.2 The System Log

The system log is supported by the **syslogd** program.

Format

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 local area network), the word `sendmail:`, and a message.

Levels

If you have **syslogd** or an equivalent installed, you will be able to do logging. There is a large amount of information that 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 ten are considered *useful*; log levels above ten are usually for debugging purposes.

A complete description of the log levels is given in section D.5.6.

D.3.3 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.

Printing the queue

The contents of the queue can be printed using the **mailq** command (or by specifying the **–bp** flag to sendmail): This command will produce a listing of the queue id's, the size of the message, the date the message entered the queue, and the sender and recipients.

Format of queue files

All queue files have the form *xfAA99999* where *AA99999* is the *id* for this file and the *x* is a type. The types are:

d	The data file. The message body (excluding the header) is kept in this file.
l	The lock file. If this file exists, the job is currently being processed, and a queue run will not process the file. For that reason, an extraneous *lf* file can cause a job to apparently disappear (it will not even time out!).
n	This file is created when an id is being created. It is a separate file to insure 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	The queue control file. This file contains the information necessary to process the job.
t	A temporary file. These are an image of the *qf* file when it is being rebuilt. It should be renamed to a *qf* file very quickly.
x	A 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	The name of the data file. There may only be one of these lines.
H	A header definition. There may be any number of these lines. The order is important: they represent the order in the final message. These use the same syntax as header definitions in the configuration file.
R	A recipient address. This will normally be completely aliased, but is actually realiased when the job is processed. There will be one line for each recipient.
S	The sender address. There may only be one of these lines.
E	An error address. If any such lines exist, they represent the addresses that should receive error messages.
T	The job creation time. This is used to compute when to time out the job.
P	The current message priority. This is used to order 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 A 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 `mckusick@calder` and `wnj`:

```
DdfA13557
Seric
T404261372
P132
Rmckusick@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.EDU>
Hreceived: by UCBARPA.BERKELEY.EDU (3.227 [10/22/82])
    id A13557; 23-Oct-82 15:49:32-PDT (Sat)
HTo: mckusick@calder, wnj
```

The example above 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

Sendmail should run the queue automatically at intervals. The algorithm is to read and sort the queue, and 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 insure that only one queue processor exists at any time, since 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 a major host going down for a couple of days may create a prohibitively large queue. This will result in Sendmail spending 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 755 mqueue
```

You should then kill the existing daemon (since it will still be 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 to just run every job in the queue. If you want to observe you can use the **–v** flag to watch what is going on.

When the queue is finally emptied, you can remove the directory:

```
rmdir /usr/spool/omqueue
```

D.3.4 The Alias Database

The alias database exists in two forms. One is a text form, maintained in the file */usr/lib/aliases*. The aliases are of the form:

```
name: name1, name2, ...
```

Only local names may be aliased; e.g.,

```
eric@mit-xx: eric@berkeley.EDU
```

will not have the desired effect. Aliases may be continued by starting any continuation lines with a space or a tab. Blank lines and lines beginning with a sharp sign (#) are comments.

The second form is processed by the **dbm** library. This form is in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag*. This is the form that Sendmail actually uses to resolve aliases. This technique is used to improve performance.

Rebuilding the alias database

The DBM version of the database may 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 will rebuild the alias database automatically if possible when it is out of date. The conditions under which it will do this are:

- The DBM version of the database is mode 666. -or-
- 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 will start the rebuild process simultaneously.

Potential problems

There are a number of problems that can occur with the alias database. They all result from a Sendmail process accessing the DBM version while it is only partially built. This can happen under two circumstances: 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.

Sendmail has 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 leaving a partially rebuilt database. Second, at the end of the rebuild it adds an alias of the form:

```
@: @
```

(which is not normally legal). Before sendmail will access the database, it checks to insure that this entry exists[1]. Sendmail will wait for this entry to appear, at which point it will force a rebuild itself[2].

List owners

If an error occurs on sending to a certain address, say f5f2x, Sendmail will look 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
```

---

[1] The **a** option is required in the configuration for this action to occur. This should normally be specified unless you are running *delivermail* in parallel with Sendmail.

[2] Note: the **D** option must be specified in the configuration file for this operation to occur. If the **D** option is not specified, a warning message is generated and Sendmail continues.

would cause `eric@ucbarpa` to get the error that will occur when someone sends to unix-wizards due to the inclusion of `nosuchuser` on the list.

D.3.5 Per-User Forwarding (.forward Files)

As an alternative to the alias database, any user may put a file with the name 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, if the home directory for user `mckusick` has a *.forward* file with contents:

```
mckusick@ernie
kirk@calder
```

then any mail arriving for `mckusick` will be redirected to the specified accounts.

D.3.6 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 builtins are described here.

Return-Receipt-To:

If this header is sent, a message will be sent to any specified addresses when the final delivery is complete, 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 will cause 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 will add 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.

At least one recipient line is required under RFC 822.

D.3.7 Sending 8-bit mail

Some sites may require that mail not be stripped of the eighth bit. Many foreign characters need the eighth bit to be accurately encoded, and neither cleared nor set arbitrarily. Because of specific requirements for SMTP, it is not currently possible to guarantee the eighth bit will be undisturbed when sending via that channel. It is, however, possible to allow it to work with other channels, notably the local channel.

The default configuration should provide this functionality. Two important things to note are a) the mailer specification (see section D.6 and section D.9) should *not* include the **L** flag, since this flag will cause the eighth bit to be stripped on all characters before they are passed to the mailer, and b) the mailer itself should be capable of properly handling 8 bit data.

The local mailer has been modified and tested to provide this support. The uucp channel is also able to handle 8 bit data. In addition, complete end-to-end reliability for unedited delivery will depend heavily on whether the receiving site strips the 8th bit during local delivery.

D.4 Arguments

The complete list of arguments to Sendmail is described in detail in Section D.7. Some important arguments are described here.

D.4.1 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, since it will only be relevant when a host that was down comes back up. If you run in **q** mode it should be relatively short, since it defines the maximum amount of time that a message may sit in the queue.

D.4.2 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
```

D.4.3 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
```

D.4.4 Debugging

There are a fairly large number of debug flags built into Sendmail . Each debug flag has a number and a level, where higher levels means to print out more information. The convention is that levels greater than nine are "absurd", i.e., they print out so much information that you wouldn't 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
```

where 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
```

For a complete list of the available debug flags you will have to look at the code (they are too dynamic to keep this documentation up to date).

D.4.5 Trying a Different Configuration File

An alternative configuration file can be specified using the **–C** flag; for example,

```
/usr/lib/sendmail -Ctest.cf
```

uses the configuration file *test.cf* instead of the default */usr/lib/sendmail.cf.* If the **–C** flag has no value it defaults to *sendmail.cf* in the current directory.

D.4.6 Changing the Values of Options

Options can be overridden using the **–o** flag. For example,

```
/usr/lib/sendmail -oT2m
```

sets the **T** (timeout) option to two minutes for this run only.

D.5 Sendmail Tuning

Several configuration parameters can be changed, 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.

D.5.1 Timeouts

All time intervals are set using a scaled syntax. For example, `10`**m** represents ten minutes, whereas `2`**h**`30`**m** represents two and a half hours.

The following list describes the full scale syntax:

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 insures that at least the sender is aware of the inability to send a message. 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
```

runs the queue and flush anything that is one day old.

D.5.2 Forking During Queue Runs

By setting the **Y** option, *sendmail* forks before each individual message while running the queue. This prevents *sendmail* from consuming large amounts of memory, 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. This can improve performance dramatically.

D.5.3 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 times the "work class factor" and the number of recipients times 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 will be 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. Since the number of recipients affects the amount of load a message presents to the system, this is also included into the priority.

The recipient and class factors can be set in the configuration file 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:

$$\text{pri} = \text{size} - (\text{class} * \text{z}) + (\text{nrcpt} * \text{y})$$

(Remember, higher values for this parameter actually mean that the job will be 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 will tend to fail again in the future.

D.5.4 Load Limiting

Sendmail can be asked to queue (but not deliver) mail if the system load average gets too high using the **x** option. When the load average exceeds the value of the **x** option, the delivery mode is set to **q** (queue only) if the *Queue Factor* (**q option**) divided by the difference in the current load average and the **x** option plus one exceeds the priority of the message — that is, the message is queued if:

$$pri > \frac{QF}{LA - x + 1}$$

The **q** option defaults to 10000, so each point of load average is worth 10000 priority points (as described above, that is, bytes + seconds + offsets).

For drastic cases, the **X** option defines a load average at which sendmail will refuse to accept network connections. Locally generated mail (including incoming UUCP mail) is still accepted.

D.5.5 Delivery Mode

Sendmail can operate in a number of delivery modes. These modes specify how quickly mail will be delivered. The **d** configuration option specifies the delivery modes. Legal modes are:

i	deliver interactively (synchronously)
b	deliver in background (asynchronously)
q	queue only (don't deliver)

There are tradeoffs. Mode **i** passes the maximum amount of information to the sender, but is hardly ever necessary. Mode **q** puts the minimum load on your machine, but means that delivery may be delayed. Mode **b** is probably a good compromise. However, this mode can cause a large number of processes if you have a mailer that takes a long time to deliver a message.

D.5.6 Log Level

The level of logging can be set for sendmail. The default using a standard configuration table is level 9. The levels are as follows:

0	No logging.
1	Major problems only.
2	Message collections and failed deliveries.
3	Successful deliveries.
4	Messages being deferred (due to a host being down, etc.).
5	Normal message queueups.
6	Unusual but benign incidents, e.g., trying to process a locked queue file.

9 Log internal queue id to external message id mappings. This can be useful for tracing a message as it travels between several hosts.

12 Several messages that are basically only of interest when debugging.

16 Verbose information regarding the queue.

D.5.7 File Modes

Many files can have a number of modes. The modes depend on what functionality you want and the level of security you require.

To suid or not suid

Sendmail can safely be made setuid to root. At the point where it is about to **exec** a mailer, it checks to see if the userid is zero; if so, it resets the userid 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 will cause mail processing to be accounted (using **sa**) to root rather than to the user sending the mail.

Should my alias database be writable?

At Berkeley we have the alias database (*/usr/lib/aliases**) mode 644. While this is not as flexible as if the database were more 666, it avoids potential security problems with a globally writable database.

The database that Sendmail actually used is represented by the two files *aliases.dir* and *aliases.pag* (both in */usr/lib*). The mode on these files should match the mode on */usr/lib/aliases* If *aliases* is writable and the DBM files (*aliases.dir and aliases.pag*) are not, users will be unable to 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 with the **newaliases** utility. If this step is ignored or forgotten any intended changes will also be ignored or forgotten.

D.6 The Whole Scoop on The Configuration File

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, since this is done every time Sendmail starts up, rather than easy for a human to read or write. On the "future project" list is a configuration-file compiler.

An overview of the configuration file is given first, followed by details of the semantics.

D.6.1 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 sharp symbol ('#') are comments.

R and S Rewriting Rules

Rewriting tools form the core of address parsing. These are an ordered production system. Sendmail 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 **S** command has the following form:

S*n*

This command sets the current ruleset being collected to *n*. If you begin a ruleset more than once it deletes the old definition.

The **R** command has the following form:

R*lhs rhs comments*

The fields must be separated by at least one tab character and there may be embedded spaces in the fields. The *lhs* field is a pattern that is applied to the input. If it matches, the input is rewritten to the *rhs* field. *Comments* are ignored.

D— define macro

Macros are named with a single character. These may be selected from the entire ASCII set, but user-defined macros should be selected from the set of upper case letters only. Lower case letters and special symbols are used internally.

Macro definitions have the following syntax:

D*x val*

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, where a *word* is a sequence of characters that do not contain characters in the **$o** macro. For example, a class of all local names for a site might be created so that attempts to send to oneself can be eliminated. These local names can either be defined directly in the configuration file or read in from another file. Classes may be given names from the set of upper case letters. Lower case letters and special characters are reserved for system use.

The C and F classes have the following syntax:

C*c word1 word2*
F*c file*

The C form defines the class *c* to match any of the named *words*. It is permissible to split the named words among multiple lines. For example, the C form:

CH`monet ucbmonet`

is equivalent to the C form:

CH`monet`
CH`ucbmonet`

The second form reads the elements of the class *c* from the named *file*.

M— define mailer

The M line defines programs and interfaces to mailers. The M line has the following format:

M*name, {field=value}**

name is the name of the mailer (used internally only). The *field=name* pairs define attributes of the mailer. The following list defines the fields:

Path	The pathname of the mailer
Flags	Special flags for the mailer
Sender	A rewriting set for sender addresses
Recipient	A rewriting set for recipient addresses
Argv	An argument vector to pass to the mailer
Eol	The end-of-line string for the mailer
Maxsize	The maximum message length to the mailer

Only the first character of the field name is checked.

H— define header

The **H** line defines the format of the header lines that Sendmail inserts into the message. The **H** has the following syntax:

H[?*mflags*?]*hname*: *htemplate*

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 the header to be automatically output. An input header is reflected to the output regardless of these flags.

O— set option

A number of *random* options can be set from a configuration file. These options are represented by single characters as shown in the following line:

O*o value*

The above option *o* is set to *value*. Depending on the option, *value* may be a string, an integer, a boolean (with legal values t, T, F, or F with TRUE as the default), or a time interval.

T— define trusted users

Trusted users are those users who are permitted to override the sender address using the **–f** flag. Trusted users are typically *root, uucp*, and *network*. It may be convenient to extend this list to include other users, perhaps to support a separate UUCP login for each host.

The trusted users line has the following syntax:

T*user1 user2...*

There can be more than one trusted user line.

P — precedence definitions

The **P** control line defines the precedence value. The **P** control line has the following syntax:

P*name=num*

A *name* found in the precedence field, is set to *num*. Higher numbers mean higher precedence. Numbers less than zero have the special property that error messages will not be returned. The default precedence is zero.

D.6.2 The Semantics

This section describes the semantics of the configuration file.

Special Macros and Conditionals

Macros are interpolated using the construct **$***x* where *x* is the name of the macro to be interpolated. In particular, lower case letters are reserved for special semantics, used to pass information in or out of Sendmail, Some special characters are reserved to provide conditionals.

The following syntax shows how conditionals can be specified:

$?*x text1* **$|** *text2* **$.**

With the **$x** macro set, the above line interpolates *text1* Otherwise the line interpolates *text2*. The following macros must be defined to transmit information into Sendmail:

e The SMTP entry message

j	The official domain name for this site
l	The format of the UNIX from line
n	The name of the daemon (for error messages)
o	The 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 **$n** macros can be considered constants except under very unusual circumstances. The **$o** macro consists of a list of characters that will be considered tokens and will separate tokens when doing parsing. For example, inserting @ in the **$o** macro, causes a@b to be scanned as three tokens: a, @, and b.

Finally, the **$q** macro specifies how an address should appear in a message.

The following macros are defined by Sendmail for interpolation in argv's for mailers or for other contexts:

a	The origination date in RFC 822 format
b	The current date in RFC 822 format
c	The hop count
d	The date in UNIX (ctime) format
f	The sender (from) address
g	The sender address relative to the recipient
h	The recipient host
i	The queue id
p	Sendmail's pid
r	Protocol used
s	Sender's host name
t	A numeric representation of the current time
u	The recipient user
v	The version number of Sendmail
w	The hostname of this site
x	The full name of the sender
z	The home directory of the recipient

Three types of dates can be used. The **$a** and **$b** macros are in RFC 822 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, **$a** is set to the current time. The **$d** macro is equivalent to the **$a** macro in UNIX (ctime) format.

The **$f** macro is the id of the sender. When mailing to a specific host, the **$g** macro is set to the address of the sender relative to the recipient.

The **$x** macro is set to the full name of the sender. This can be determined in several ways. It can be passed as 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 **$z** macros get set to the host, user, and home directory (if local) of the recipient. The first two macros are set from the **$@** and **$:** part of the rewriting rules, respectively.

The **$p** and **$t** macros are used to create unique strings (e.g., for the *Message-Id:* field). The **$i** macro is set to the queue id on the host. If the macro is put into the timestamp line it can be extremely useful for tracking messages. The **$v** macro is set to Sendmail version number. This macor is normally put in timestamps is useful for debugging. The **$w** macro is set to the host name if it can be determined. The **$c** field is set to the hop count, The hop count can be determined by the **–h** flag on the command line or by counting the timestamps in the message.

Special Classes

The **$=w** class is set to all known host names. This class can be used to match local hostnames.

The left hand side

The left hand side of rewriting rules contains a pattern where normal words are simply matched directly. Metasyntax is introduced using a dollar sign as shown in the following list:

$*	Match zero or more tokens
$+	Match one or more tokens
$–	Match exactly one token
$=1*x*	Match any token in class *x*
$˜*x*	Match any token not in class *x*

If any of metasymbols match, they are assigned to the symbol $ for replacement on the right hand side, *n* is the index in the LHS.

The right hand side

When the left hand side (LHS) of a rewriting rule matches, the input is deleted and replaced by the right hand side (RHS) Tokens are copied directly from the RHS. unless they begin with a dollar sign. The following list describes the RHS metasymbols:

$*n*	Substitute indefinite token *n* from LHS
$[***name***$]	Canonicalize *name*
$>*n*	Call ruleset n **$#***mailer* Resolve to *mailer* **$@***host* Specify *host* **$:***user* Specify *user*

The **$***n* syntax substitutes the corresponding value from a **$+**, **$–**, **$***, **$=**, or **$~** match on the LHS and may be used anywhere.

A host name enclosed between **$[** and **$]** is looked up using the *gethostent*(3) routines and replaced by the canonical name.

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. This syntax causes evaluation of the ruleset to terminate immediately, and signals to Sendmail that the address has completely resolved. The complete syntax is:

$#***mailer***$@*host***$:***user*

This syntax specifies the three (mailer, host, user) parameters necessary to direct the mailer. If the mailer is local the host part may be omitted. The *mailer* and *host* must be a single word, but the *user* may be multi-part.

A RHS may 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 be used to avoid continued application of a rule. The prefix is stripped before continuing.

The **$@** and **$:** prefixes may precede a **$>** specification.

Substitution occurs in the following order: parameters from the LHS are substituted, hostnames are canonicalized, subroutines are called, and finally **$#**, **$@**, and **$:** are processed.

Semantics of Rewriting Rule Sets

Sendmail contains five rewriting rule sets. These are related as depicted by Figure 1.

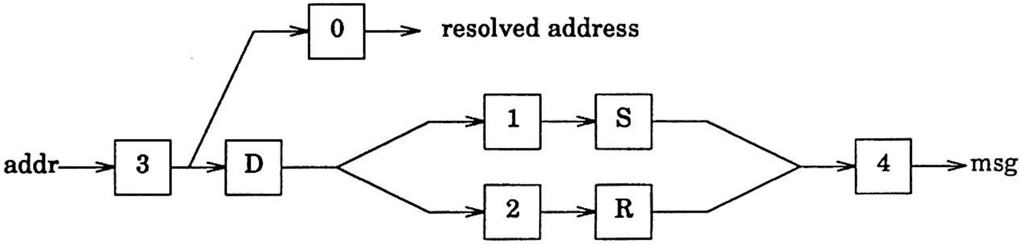


Figure 1 — Rewriting set semantics

D — sender domain addition
S — mailer-specific sender rewriting
R — mailer-specific recipient rewriting

Ruleset three should turn the address into canonical form. The canonical form should have the following basic syntax:

local-part@host-domain-spec

If no @ sign is specified, then the host-domain-spec may be appended from the sender address (if the **C** flag is set in the mailer definition corresponding to the *sending* mailer). Sendmail applies ruleset three before doing anything with any address.

Ruleset zero is applied after ruleset three to addresses that 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

Several flags may be associated with each mailer. A letter of the alphabet identifies each mailer. Many of them are assigned semantics internally.

The error mailer

The *error* mailer can be used to 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, the entry:

```
$#error$:Host unknown in this domain
```

on the RHS of a rule causes the specified error to be generated if the LHS matches. The error mailer is only functional in ruleset zero.

D.6.3 Building a Configuration File From Scratch

Building a configuration table 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 it is that you are trying to do and come up with a philosophy for the configuration table. This section is intended to explain what the real purpose of a configuration table is and to give you some ideas for what your philosophy might be.

What you are trying to do

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, etc. Since these are described in other places, 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. Sendmail does this in three subphases. Rulesets one and two are applied to all sender and recipient addresses respectively. After this, you may 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 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. The mailer descriptor describes the interface requirements of the mailer.

Philosophy

The particular philosophy you choose will depend heavily on the size and structure of your organization. I will present a few possible philosophies here.

One general point applies to all of these philosophies: it is almost always a mistake to try to do full name resolution. For example, if you are trying to get names of the form `user@host` to the Arpanet, it does not pay to route them to `xyzvax!decvax!ucbvax!c70:user@host` since you then depend on several links not under your control. The best approach to this problem is to simply forward to `xyzvax!user@host` and let xyzvax worry about it from there. In summary, just get the message closer to the destination, rather than determining the full path.

Large site, many hosts — minimum information. Berkeley is an example of a large site, i.e., more than two or three hosts and multiple mail connections. We have decided that the only reasonable philosophy in our environment is to designate one host as the guru for our site. It must be able to resolve any piece of mail it receives. The other sites should have the minimum amount of information they can get away with. In addition, any information they do have should be hints rather than solid information.

For example, a typical site on our local ether network is `monet`. When monet receives mail for delivery, it checks whether it knows that the destination host is directly reachable; if so, mail is sent to that host. If it receives mail for any unknown host, it just passes it directly to *ucbvax,* our master host. Ucbvax may determine that the host name is illegal and reject the message, or may be able to do delivery. However, it is important to note that when a new mail connection is added, the only host that *must* have its tables updated is ucbvax; the others *may* be updated if convenient, but this is not critical.

This picture is slightly muddied due to network connections that are not actually located on ucbvax. For example, some UUCP connections are currently on *ucbarpa*. However, monet "does not" know about this; the information is hidden totally between ucbvax and ucbarpa. Mail going from monet to a UUCP host is transferred via the ethernet from monet to ucbvax, then via the ethernet from ucbvax to ucbarpa, and then is submitted to UUCP. Although this involves some extra hops, we feel this is an acceptable tradeoff.

An interesting point is that it would be possible to update monet to send appropriate UUCP mail directly to ucbarpa if the load got too high; if monet failed to note a host as connected to ucbarpa it would go via ucbvax as before, and if monet incorrectly sent a message to ucbarpa it would still be sent by ucbarpa to ucbvax as before. The only problem that can occur is loops, for example, if ucbarpa thought that ucbvax had the UUCP connection and vice versa. For this reason, updates should *always* happen to the master host first.

This philosophy results as much from the need to have a single source for the configuration files (typically built using **m4** or some similar tool) as any logical need. Maintaining more than three separate tables by hand is essentially an impossible job.

Small site — complete information. A small site (two or three hosts and few external connections) may find it more reasonable to have complete information at each host. This would require that each host know exactly where each network connection is, possibly including the names of each host on that network. As long as the site remains small and the the configuration remains relatively static, the update problem will probably not be too great.

Single host. This is in some sense the trivial case. The only major issue is trying to insure that you don't have to know too much about your environment. For example, if you have a UUCP connection you might find it useful to know about the names of hosts connected directly to you, but this is really not necessary since this may be determined from the syntax.

Relevant issues

The canonical form you use should almost certainly be as specified in the Arpanet protocols RFC819 and RFC822. Copies of these RFC's are included on the Sendmail tape as *doc/rfc819.lpr* and *doc/rfc822.lpr*

RFC822 describes the format of the mail message itself. Sendmail follows this RFC closely, to the extent that many of the standards described in this document can not 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 which 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 Berkeley one legal host might be `a.CC.Berkeley.EDU` ; reading from right to left, `EDU` is a top level domain comprising educational institutions, `Berkeley` 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.

Beware when reading RFC819 that there are a number of errors in it.

How to proceed

Once you have decided on a philosophy, it is worth examining the available configuration tables to decide if any of them are close enough to steal major parts of. Even under the worst of conditions, there is a fair amount of boiler plate that can be collected safely.

The next step is to build ruleset three. This will be the hardest part of the job. Beware of doing too much to the address in this ruleset, since anything you do will reflect through to the message. In particular, stripping of local domains is best deferred, since this can leave you with addresses with no domain spec at all. Since 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 Berkeley 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
```

which would read the configuration file *test.cf* and enter test mode.

In this mode, you enter lines of the form:

```
rwset address
```

where `rwset` is the rewriting set you want to use and `address` is an address to apply the set to. Test mode shows you the steps it takes as it proceeds, finally showing you the address it ends up with. 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
```

first applies ruleset three to the input `monet:bollard.` Ruleset one is then applied to the output of ruleset three, followed similarly by rulesets twenty-one and four.

If you need more detail, you can also use the **–d21** flag to turn on more debugging. For example,

```
sendmail -bt -d21.99
```

turns on an incredible amount of information; a single word address is probably going to print out several pages worth of information.

Building mailer descriptions

To add an outgoing mailer to your mail system, you will have to 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 flag. You should specify an **k** flag to pass the name of the sender as a **–k** flag. This flag is passed only if it was passed to Sendmail, so that mailers that give errors under some circumstances can be placated. If the mailer must be called as **root** the **S** flag should be given; this will not reset the userid before calling the mailer[3]. If this mailer is local (i.e., will perform 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** will be repeated for each unique user on a given host. The **e** flag will mark the mailer as being *expensive*, which will cause

---

[3] Sendmail must be running setuid to root for this to work.

Sendmail to defer connection until a queue run[4]. 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 spec of the sender (i.e., the `@host.domain` part) is saved and is appended to any addresses in the message that do not already contain a domain spec. For example, a message of the form:

```
From: eric@ucbarpa
To: wnj@monet, mckusick
```

will be modified to:

```
From: eric@ucbarpa
To: wnj@monet, mckusick@ucbarpa
```

if and only if the **C** flag is defined in the mailer corresponding to `eric@ucbarpa`.

Other flags are described later.

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, a header of the form:

```
From: eric
```

might be changed to be:

```
From: eric@ucbarpa
```

or

```
From: ucbvax!eric
```

depending on the domain it is being shipped into. 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 E field. It may have embedded spaces. If there is no argv with a **$u** macro in it, Sendmail will speak SMTP to the mailer. If the pathname for this mailer is `[IPC]`, the argv should be

```
IPC $h [ port ]
```

---

[4] The c configuration option must be given for this to be effective.

where `port` is the optional port number to connect to.

For example, the specifications:

```
Mlocal,   P=/bin/mail,     F=rlsm  S=10,   R=20,   A=mail -d $u
Mether,   P=[IPC],         F=meC,  S=11,   R=21,   A=IPC $h, M=100000
```

specifies a mailer to do local delivery and a mailer for ethernet delivery. The first is called `local`, is located in the file */bin/mail*, takes a picky **–k** flag, does local delivery, quotes should be stripped from addresses, and multiple users can be delivered at once; ruleset ten should be applied to sender addresses in the message and ruleset twenty should be applied to recipient addresses; the argv to send to a message will be the word **mail**, the word **–d**, and words containing the name of the receiving user. If a **–k** flag is inserted it will be between the words `mail` and `-d`. The second mailer is called `ether`, it should be connected to 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 eleven and recipient addresses by ruleset twenty-one. There is a 100,000 byte limit on messages passed through this mailer.

D.7 Command Line Flags

Arguments must be presented with flags before addresses. The flags are:

–k *addr*	The sender's machine address is *addr*. This flag is ignored unless the real user is listed as a "trusted user" or if *addr* contains an exclamation point (because of certain restrictions in UUCP).
–h *cnt*	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 17) Sendmail throws away the message with an error.
–F*name*	Sets the full name of this user to *name*.
–n	Don't do aliasing or forwarding.
–t	Read the header for `To:`, `Cc:`, and `Bcc:` lines, and send to everyone listed in those lists. The `Bcc:` line will be deleted before sending. Any addresses in the argument vector will be deleted from the send list.

–b*x*	Set operation mode to *x*. 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, don't 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 [Neigus73, Postel74, Postel77]), and ending lines of error messages with <CRLF>.
–q*time*	Try to process the queued up mail. If the time is given, a sendmail will run through the queue at the specified interval to deliver queued mail; otherwise, it only runs once.
–C*file*	Use a different configuration *file*. Sendmail runs as the invoking user (rather than root) when this flag is specified.
–d*level*	Set debugging level.
–o*xvalue*	Set option *x* to the specified *value*. These options are described in following section

There are a number of options that may be specified as primitive flags (provided for compatibility with *delivermail*). These are the e, i, m, and v options. Also, the f option may be specified as the –s flag.

D.8 Configuration Options

The following options may 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, don't connect immediately. This requires that queueing be compiled in, since it will depend 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 stat always
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 in to *n*.
H*file*	Specify the help *file* for SMTP.
I	Insist that the BIND name server be running to resolve host names. If this is not set and the name server is not running, the */etc/hosts* file will be considered complete. In general, you do want to set this option if your */etc/hosts* file does not include all hosts known to you or if you are using the MX (mail forwarding)

	feature of the BIND name server. The name server will still be consulted even if this option is not set, but Sendmail will feel free to resort to reading */etc/hosts* if the name server is not available. Thus, you should *never* set this option if you do not run the name server.
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 by default. The 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, i.e., spaces delimit names. This actually turns on an adaptive algorithm: if any recipient address contains a comma, parenthesis, or angle bracket, it will be 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.
qfactor	Use *factor* as the multiplier in the map function to decide when to just 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 will be sent. Defaults to 10000.
rtime	Timeout reads after *time* interval.
S*file*	Log statistics in the named *file*.
s	Be super-safe when running things, i.e., always instantiate the queue file, even if you are going to attempt immediate delivery. Sendmail always instantiates the queue file before returning control the the client under any circumstances.

T*time*	Set the queue timeout to *time*. After this interval, messages that have not been successfully sent will be returned to the sender.
t*S*,*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 userid for mailers to *n*. Mailers without the *S* flag in the mailer definition will run as this user.
v	Run in verbose mode.
x*LA*	When the system load average exceeds *LA*, just queue messages (i.e., don't try to send them).
X*LA*	When the system load average exceeds *LA*, refuse incoming SMTP connections.
y*fact*	The indicated *fact* is added to the priority (thus *lowering* the priority of the job) for each recipient, i.e., 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, since the default tends to consume considerable amounts of memory while the queue is being processed.
z*fact*	The indicated *fact* 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: will be favored.
Z*fact*	The *fact* is added to the priority every time a job is processed. Thus, each time a job is processed, its priority will be decreased by the indicated value. In most environments this should be positive, since hosts that are down are all too often down for a long time.

D.9 Mailer Flags

The following flags may be set in the mailer description.

f	The mailer wants a **–f** `from` flag, but only if this is a network forward operation (i.e., the mailer will give an error if the executing user does not have special permissions).
k	Same as **f**, but sends a **–k** flag.
S	Don't reset the userid 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 (e.g, a user's mail.cf file).
n	Do not insert a UNIX-style `From` line on the front of the message.
l	This mailer is local (i.e., final delivery will be performed).
s	Strip quote characters off of 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 will be 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	Upper case should be preserved in user names for this mailer.
h	Upper case 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-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 will occur during a queue run.

X This mailer want to use the hidden dot algorithm as specified in RFC821; basically, any line beginning with a dot will have an extra dot prepended (to be stripped at the other end). This insures that lines in the message containing a dot will 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 will be speaking SMTP to another Sendmail – as such it can use special protocol features. Specifying this flag will also cause the eighth bit to be stripped from all the characters in the message before it is passed to the mailer. This option is not required (i.e., if this option is omitted the transmission will still operate 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 will have the `@domain` clause from the sender tacked on. This allows mail with headers of the form:

```
From: usera@hosta To: userb@hostb, userc
```

to be rewritten automatically as:

```
From: usera@hosta To: userb@hostb, userc@hosta
```

E Escape lines beginning with `From` in the message with a '>' sign.

D.10 Other Configuration

There are some configuration changes that can be made by recompiling Sendmail. These are located in two places:

src/conf.h	Configuration parameters that may be tweaked by the installer are included in conf.h.

src/conf.c	Some special routines and a few variables may be defined in conf.c. For the most part these are selected from the settings in *conf.h*.

Parameters in src/conf.h

Parameters and compilation options are defined in *conf.h*. Most of these need not normally be tweaked; common parameters are all in sendmail.cf. However, the sizes of certain primitive vectors, etc., are included in this file. The numbers following the parameters are their default value.

MAXLINE [1024]	The maximum line length of any input line. If message lines exceed this length they will still be processed correctly; however, header lines, configuration file lines, alias lines, etc., must fit within this limit.
MAXNAME [256]	The maximum length of any name, such as a host or a user name.
MAXFIELD [2500]	The maximum total length of any header field, including continuation lines.
MAXPV [40]	The maximum number of parameters to any mailer. This limits the number of recipients that may be passed in one transaction.
MAXHOP [17]	When a message has been processed more than this number of times, sendmail rejects the message on the assumption that there has been an aliasing loop. This can be determined from the **–h** flag or by counting the number of trace fields (i.e, `Received:` lines) in the message header.
MAXATOM [100]	The maximum number of atoms (tokens) in a single address. For example, the address "eric@Berkeley" is three atoms.
MAXMAILERS [25]	The maximum number of mailers that may be defined in the configuration file.

MAXRWSETS [30]

The maximum number of rewriting sets that may be defined.

MAXPRIORITIES [25]

The maximum number of values for the `Precedence:` field that may be defined (using the **P** line in sendmail.cf).

MAXTRUST [30]

The maximum number of trusted users that may be defined (using the **T** line in sendmail.cf).

MAXUSERENVIRON [40]

The maximum number of items in the user environment that will be passed to subordinate mailers.

QUEUESIZE [600]

The maximum number of entries that will be processed in a single queue run.

A number of other compilation options exist. These specify whether or not specific code should be compiled in.

DBM	If set, the DBM package in UNIX is used (see **dbm** in [UNIX80]). If not set, a much less efficient algorithm for processing aliases is used.
NDBM	If set, the new version of the DBM library that allows multiple databases will be used. DBM must also be set.
DEBUG	If set, debugging information is compiled in. To actually get the debugging output, the **–d** flag must be used.
LOG	If set, the **syslog** routine in use at some sites is used. This makes an informational log record for each message processed, and makes a higher priority log record for internal system errors.
QUEUE	This flag should be set to compile in the queueing code. If this is not set, mailers must accept the mail immediately or it will be returned to the sender.
SMTP	If set, the code to handle user and server SMTP will be compiled in. This is only necessary if your machine has some mailer that speaks SMTP.

DAEMON	If set, code to run a daemon is compiled in. This code is for 4.2 or 4.3BSD.
UGLYUUCP	If you have a UUCP host adjacent to you which is not running a reasonable version of *rmail*, you will have to set this flag to include the `remote from sysname` info on the from line. Otherwise, UUCP gets confused about where the mail came from.
NOTUNIX	If you are using a non-UNIX mail format, you can set this flag to turn off special processing of UNIX-style `From` lines.
NAMED_BIND	Compile in code to use the Berkeley Internet Name Domain (BIND) server to resolve TCP/IP host names.
SETPROCTITLE	If defined, Sendmail will change its *argv* array to indicate its current status. This can be used in conjunction with the command to find out just what it's up to.
NO_WILDCARD_MX	Should be set if there are no wildcard MX nameserver records in the local domain. If set, this will enable the use of ANY query types, resulting in better performance. Unfortunately, wildcard MX records in the local domain will mess this up, hence the need for this compilation option.

Configuration in src/conf.c

Not all header semantics are defined in the configuration file. Header lines that should only be included by certain mailers (as well as other more obscure semantics) must be specified in the *HdrInfo* table in *conf.c*. This table contains the header name (which should be in all lower case) and a set of header control flags (described below), The flags are:

H_ACHECK

Normally when the check is made to see if a header line is compatible with a mailer, Sendmail will not delete an existing line. If this flag is set, Sendmail will delete even existing header lines. That is, if this bit is set and the mailer does not have flag bits set that intersect with the required mailer flags in the header definition in sendmail.cf, the header line is *always* deleted.

H_EOH If this header field is set, treat it like a blank line, i.e., it will signal the end of the header and the beginning of the message text.

H_FORCE
Add this header entry even if one existed in the message before. If a header entry does not have this bit set, Sendmail will not add another header line if a header line of this name already existed. This would normally be used to stamp the message by everyone who handled it.

H_TRACE
If set, this is a timestamp (trace) field. If the number of trace fields in a message exceeds a preset amount the message is returned on the assumption that it has an aliasing loop.

H_RCPT If set, this field contains recipient addresses. This is used by the –t flag to determine who to send to when it is collecting recipients from the message.

H_FROM This flag indicates that this field specifies a sender. The order of these fields in the *HdrInfo* table specifies Sendmail's preference for which field to return error messages to.

Let's look at a sample *HdrInfo* specification:

```
struct hdrinfo   HdrInfo[] =
{
             /* originator fields, most to least significant  */
        "resent-sender",        H_FROM,
        "resent-from",          H_FROM,
        "sender",               H_FROM,
        "from",                 H_FROM,
        "full-name",            H_ACHECK,
             /* destination fields */
        "to",                   H_RCPT,
        "resent-to",            H_RCPT,
        "cc",                   H_RCPT,
             /* message identification and control */
        "message",              H_EOH,
        "text",                 H_EOH,
             /* trace fields */
        "received",             H_TRACE|H_FORCE,

        NULL,                   0,
};
```

This structure indicates that the `To:`, `Resent-To:`, and `Cc:` fields all specify recipient addresses. Any `Full-Name:` field will be deleted unless the required mailer flag (indicated in the configuration file) is specified. The `Message:` and `Text:` fields will terminate the header; these are specified in new protocols [NBS80] or used by random dissenters around the network world. The `Received:` field will always be added, and can be used to trace messages.

There are a number of important points here. First, header fields are not added automatically just because they are in the *HdrInfo* structure; they must be specified in the configuration file in order to be added to the message. Any header fields mentioned in the configuration file but not mentioned in the *HdrInfo* structure have default processing performed; that is, they are added unless they were in the message already. Second, the *HdrInfo* structure only specifies cliched processing; certain headers are processed specially by ad hoc code regardless of the status specified in *HdrInfo* . For example, the `Sender:` and `From:` fields are always scanned on ARPANET mail to determine the sender; this is used to perform the "return to sender" function. The `From:` and `Full-Name:` fields are used to determine the full name of the sender if possible; this is stored in the macro **$x** and used in a number of ways.

The file *conf.c* also contains the specification of ARPANET reply codes. There are four classifications these fall into:

```
char    Arpa_Info[] =    "050";  /* arbitrary info */
char    Arpa_TSyserr[] = "455";  /* some (transient) system error */
char    Arpa_PSyserr[] = "554";  /* some (permanent) system error */
char    Arpa_Usrerr[] =  "554";  /* some (fatal) user error */
```

The class *Arpa_Info* is for any information that is not required by the protocol, such as forwarding information. *Arpa_TSyserr* and *Arpa_PSyserr* is printed by the **syserr** routine. TSyserr is printed out for transient errors, that is, errors that are likely to go away without explicit action on the part of a systems administrator. PSyserr is printed for permanent errors. The distinction is made based on the value of *errno*. Finally, *Arpa_Usrerr* is the result of a user error and is generated by the **usrerr** routine; these are generated when the user has specified something wrong, and hence the error is permanent, i.e., it will not work simply by resubmitting the request.

If it is necessary to restrict mail through a relay, the **checkcompat** routine can be modified. This routine is called for every recipient address. It can return **TRUE** to indicate that the address is acceptable and mail processing will continue, or it can return **FALSE** to reject the recipient. If it returns false, it is up to *checkcompat* to print an error message (using **usrerr**) saying why the message is rejected. For example, *checkcompat* could read:

```
bool
checkcompat(to)
        register ADDRESS *to;
{
        if (MsgSize > 50000 && to->q_mailer != LocalMailer)
        {
                usrerr("Message too large for non-local delivery"
                NoReturn = TRUE;
                return (FALSE);
        }
        return (TRUE);
}
```

This would reject messages greater than 50000 bytes unless they were local. The **NoReturn** flag can be sent to suppress the return of the actual body of the message in the error return. The actual use of this routine is highly dependent on the implementation, and use should be limited.

Configuration in src/daemon.c

The file *src/daemon.c* contains a number of routines that are dependent on the local networking environment. The version supplied is specific to 4.3 BSD.

The routine **maphostname** is called to convert strings within
$[...
$] symbols. It can be modified if you wish to provide a more sophisticated service, e.g., mapping UUCP host names to full paths.

D.11 Summary of Support Files

This is a summary of the support files that Sendmail creates or generates.

/usr/lib/sendmail	The binary of Sendmail.
/usr/bin/newaliases	A 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	The configuration file, in textual form.
/usr/lib/sendmail.fc	The configuration file represented as a memory image.
/usr/lib/sendmail.hf	The SMTP help file.
/usr/lib/sendmail.st	A statistics file; need not be present.
/usr/lib/aliases	The textual version of the alias file.
/usr/lib/aliases.{pag,dir}	The alias file in **dbm** format.
/usr/spool/mqueue	The 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**	A file used when creating a unique id.
*/usr/spool/mqueue/xf**	A transcript of the current session.

E. Sendmail
Configuration Files

E. Sendmail
Configuration Files

Appendix E
Sendmail Configuration Files

E.1 Introduction E-1
E.2 TCP/IP Configuration File E-1
E.3 UUCP Configuration File E-12
E.4 TCP/IP and UUCP Configuration Files E-22

Appendix E
Sendmail Configuration Files

E.1 Introduction

The ptx/TCP/IP software contains the following off-the-shelf Sendmail configuration files that you can modify for use at your site:

tcpproto.cf	Used on systems connected by a network that uses TCP/IP as the communication protocol.
uucpproto.cf	Used on systems connected by UUCP only.
tcpuucpproto.cf	Used on a system that connects to other systems through a TCP/IP network and and *also* connects to the outside world through UUCP.

The following sections contain each of the files listed above.

E.2 TCP/IP Configuration File

The following file should be used on systems connected by a network that uses the TCP/IP communication protocol:

```
#
# Copyright (c) 1983 Eric P. Allman
# Copyright (c) 1988 The Regents of the University of California.
# All rights reserved.
#
# Redistribution and use in source and binary forms are permitted
# provided that the above copyright notice and this paragraph are
# duplicated in all such forms and that any documentation,
# advertising materials, and other materials related to such
# distribution and use acknowledge that the software was developed
# by the University of California, Berkeley.  The name of the
# University may not be used to endorse or promote products derived
# from this software without specific prior written permission.
# THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
# IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
# WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
#
```

```
#  @(#)tcpproto.mc 1.2 (Berkeley) 1/24/89
#  @(#)proto.mc 1.21 (Berkeley) 2/15/89
#
# built by keany on Fri Dec 22 08:23:29 PST 1989
# in /crg3/comm/keany/ptx_alpha/usr/src/bin/sendmail/cf/cf on crg3
#
############################################################
############################################################
#####
#####     SENDMAIL CONFIGURATION FILE
#####
############################################################
############################################################


##################
#   local info   #
##################

# file containing our internet aliases
Fw/usr/lib/sendmail.cw



#############################
###   Setup Information   ###
#############################


######################
#   General Macros   #
######################

# local domain name
DDYOUR_DOMAIN_GOES_HERE



# UUCP relay host
DRYOUR_UUCP_RELAY_GOES_HERE

# csnet relay host
DCYOUR_CSNET_RELAY_GOES_HERE

# bitnet relay host
DBYOUR_BITNET_RELAY_GOES_HERE

# my official hostname
Dj$w
```

```
###############
#   Classes   #
###############

# Internal ("fake") domains that we use in rewriting
CIUUCP BITNET CSNET




######################
#   Version Number   #
######################

DZ1.35


######################
#   Special macros   #
######################

# my name
DnMAILER-DAEMON
# UNIX header format
DlFrom $g  $d
# delimiter (operator) characters
Do.:%@!^=/[]
# format of a total name
Dq$g$?x ($x)$.
# SMTP login message
De$j Sendmail $v/$Z ready at $b

###############
#   Options   #
###############

# location of alias file
OA/usr/lib/aliases
# wait up to ten minutes for alias file rebuild
Oa10
# substitution for space (blank) characters
OB.
# (don't) connect to "expensive" mailers
#Oc
# default delivery mode (deliver in background)
Odbackground
# temporary file mode
OF0600
# default GID
Og1
```

```
# location of help file
OH/usr/lib/sendmail.hf
# log level
OL9
# default network name
ONARPA
# default messages to old style
Oo
# queue directory
OQ/usr/spool/mqueue
# read timeout -- violates protocols
Or2h
# status file
OS/usr/lib/sendmail.st
# queue up everything before starting transmission
Os
# default timeout interval
OT3d
# time zone names (V6 only)
OtPST,PDT
# default UID
Ou1
# wizard's password
OW*
# load average at which we just queue messages
Ox8
# load average at which we refuse connections
OX12

###########################
#   Message precedences   #
###########################

Pfirst-class=0
Pspecial-delivery=100
Pbulk=-60
Pjunk=-100

#####################
#   Trusted users   #
#####################

Troot
Tdaemon
Tuucp

#########################
#   Format of headers   #
#########################
```

```
H?P?Return-Path: <$g>
HReceived: $?sfrom $s $.by $j ($v/$Z)
   id $i; $b
H?D?Resent-Date: $a
H?D?Date: $a
H?F?Resent-From: $q
H?F?From: $q
H?x?Full-Name: $x
HSubject:
# HPosted-Date: $a
# H?l?Received-Date: $b
H?M?Resent-Message-Id: <$t.$i@$j>
H?M?Message-Id: <$t.$i@$j>



###########################
###   Rewriting Rules   ###
###########################


################################
#  Sender Field Pre-rewriting  #
################################
S1
#R$*<$*>$*      $1$2$3          defocus

###################################
#  Recipient Field Pre-rewriting  #
###################################
S2
#R$*<$*>$*      $1$2$3          defocus



#################################
#  Final Output Post-rewriting  #
#################################
S4

R@         $@           handle <> error addr

# resolve numeric addresses to name if possible
R$*<@[$+]>$*    $:$1<@$[[$2]$]>$3      lookup numeric internet addr

# externalize local domain info
R$*<$+>$*    $1$2$3           defocus
R@$+:@$+:$+     @$1,@$2:$3          <route-addr> canonical

# UUCP must always be presented in old form
```

```
R$+@$-.UUCP        $2!$1              u@h.UUCP => h!u

# delete duplicate local names
R$+%$=w@$=w        $1@$w              u%host@host => u@host
R$+%$=w@$=w.$D        $1@$w              u%host@host => u@host


###########################
#  Name Canonicalization  #
###########################
S3

# handle "from:<>" special case
R$*<>$*            $@@              turn into magic token

# basic textual canonicalization -- note RFC733 heuristic here
R$*<$*<$*<$+>$*>$*>$*   $4             3-level <> nesting
R$*<$*<$+>$*>$*      $3             2-level <> nesting
R$*<$+>$*     $2             basic RFC821/822 parsing

# make sure <@a,@b,@c:user@d> syntax is easy to parse -- undone later
R@$+,$+            @$1:$2            change all "," to ":"

# localize and dispose of route-based addresses
R@$+:$+            $@$>6<@$1>:$2        handle <route-addr>

# more miscellaneous cleanup
R$+         $:$>8$1              host dependent cleanup
R$+:$*;@$+         $@$1:$2;@$3           list syntax
R$+:$*;            $@$1:$2;          list syntax
R$+@$+          $:$1<@$2>         focus on domain
R$+<$+@$+>         $1$2<@$3>         move gaze right
R$+<@$+>        $@$>6$1<@$2>      already canonical

# convert old-style addresses to a domain-based address
R$+^$+          $1!$2             convert ^ to !
R$-!$+          $@$>6$2<@$1.UUCP>       resolve uucp names
R$+.$-!$+       $@$>6$3<@$1.$2>         domain uucps
R$+!$+          $@$>6$2<@$1.UUCP>       uucp subdomains
R$+%$+          $:$>9$1%$2          user%host
R$+<@$+>        $@$>6$1<@$2>        already canonical
R$-.$+          $@$>6$2<@$1>        host.user


#################################
#   special local conversions   #
#################################

S6
R$*<@$=w>$*        $:$1<@$w>$3          get into u@$w form
```

```
R$*<@$=w.$D>$*      $:$1<@$w>$3
R$*<@$=U.UUCP>$* $:$1<@$w>$3

#################################
#   Change rightmost % to @.    #
#################################

S9
R$*%$*       $1@$2            First make them all @'s.
R$*@$*@$*    $1%$2@$3         Undo all but the last.
R$*@$*       $@$1<@$2>        Put back the brackets.



###################
###   Mailers   ###
###################


#############################################################
#############################################################
#####
#####     Local and Program Mailer specification
#####
#############################################################
#############################################################

Mlocal,   P=/bin/mail, F=rlsDFMmn, S=10, R=20, A=mail -d $u
Mprog, P=/bin/sh,   F=lsDFMe,   S=10, R=20, A=sh -c $u

S10
R@        $n        errors to mailer-daemon


#############################################################
#############################################################
#####
#####     Local Domain SMTP Mailer specification
#####
#####  Messages processed by this specification are assumed to remain
#####  the local domain.  Hence, they can refer to hosts that are
#####  not registered in the NIC host table.
#####
#############################################################
#############################################################

Mtcpld,   P=[IPC], F=mDFMueXLC, S=17, R=27, A=IPC $h, E= r n

S17
```

```
# cleanup forwarding a bit
R$*<$*>$*     $1$2$3          defocus
R$*       $:$>3$1             canonicalize
R$*%$*<@$w>     $:$>9$1%$2          user%localhost@localdomain

# pass <route-addr>'s through
R<@$+>$*      $@<@$[$1$]>$2       resolve <route-addr>

# map colons to dots everywhere
R$*:$*        $1.$2           map colons to dots

# output local host as user@host.domain
R$-        $@$1<@$w>       user w/o host
R$+<@$w>      $@$1<@$w>       this host
R$+<@$=w>     $@$1<@$w>       or an alias
R$+<@$->      $:$1<@$[$2$]>       ask nameserver
R$+<@$w>      $@$1<@$w>       this host
R$+<@$->      $@$1<@$2.$D>        if nameserver fails

# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I>     $@$1<@$[$2.$3$]>    user@host.domain
R$+<@[$+]>       $@$1<@[$2]>         already ok

# output fake domains as user%fake@relay

R$+<@$+.BITNET>     $@$1%$2.BITNET<@$B>    user@host.bitnet
R$+<@$+.CSNET>      $@$1%$2.CSNET<@$C>     user@host.CSNET
R$+<@$+.UUCP>    $@$2!$1<@$w>        user@host.UUCP


S27

# cleanup
R$*<$*>$*     $1$2$3          defocus
R$*       $:$>3$1             now canonical form
R$*%$*<@$w>     $:$>9$1%$2          user%localhost@localdomain

# pass <route-addr>'s through
R<@$+>$*      $@<@$[$1$]>$2       resolve <route-addr>

# map colons to dots everywhere
R$*:$*        $1.$2           map colons to dots

# output local host as user@host.domain
R$-        $@$1<@$w>       user w/o host
R$+<@$w>      $@$1<@$w>       this host
R$+<@$=w>     $@$1<@$w>       or an alias
R$+<@$->      $:$1<@$[$2$]>       ask nameserver
R$+<@$w>      $@$1<@$w>       this host
R$+<@$->      $@$1<@$2.$D>        if nameserver fails
```

```
# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I>    $@$1<@$[$2.$3$]>    user@host.domain
R$+<@[$+]>      $@$1<@[$2]>         already ok

# output fake domains as user%fake@relay

R$+<@$+.BITNET>     $@$1%$2.BITNET<@$B>    user@host.BITNET
R$+<@$+.CSNET>      $@$1%$2.CSNET<@$C>     user@host.CSNET
R$+<@$+.UUCP>   $@$2!$1             user@host.UUCP



############################################################
############################################################
#####
#####     Internet SMTP Mailer specification
#####
#####  Messages processed by this specification are assumed to leave
#####  the local domain -- hence, they must be canonical according to
#####  RFC822 etc.  This means that machines not registered with
#####  the NIC must be hidden behind our Internet relay.
#####
############################################################
############################################################

Mtcp,  P=[IPC], F=mDFMueXLC, S=14, R=24, A=IPC $h, E= r n

S14

# pass <route-addr>'s through
R<@$+>$*     $@<@$[$1$]>$2       resolve <route-addr>

# map colons to dots everywhere
R$*:$*       $1.$2           map colons to dots

# output local host in user@host.domain syntax
R$-       $1<@$w>            user w/o host
R$+<@$=w>    $:$1<@$w>       this host
R$+<@$->     $:$1<@$[$2$]>       canonicalize into dom
R$+<@$->     $:$1<@$2.$D>        if nameserver fails
R$+<@$=N.$D>     $@$1<@$2.$D>       nic-reg hosts are ok
R$+<@$*.$D>      $@$1%$2.$D<@$A>        else -> u%h@gateway

# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I>    $@$1<@$[$2.$3$]>    user@host.domain
R$+<@[$+]>      $@$1<@[$2]>         already ok

# output internal ("fake") domains as "user%host@relay"

R$+<@$+.BITNET>     $@$1%$2.BITNET<@$B>    user@host.BITNET
```

```
R$+<@$+.CSNET>     $@$1%$2.CSNET<@$C>     user@host.CSNET
R$+<@$+.UUCP>   $@$2!$1<@$w>       user@host.UUCP


S24

# put in <> kludge
R$*<$*>$*    $1$2$3          defocus
R$*       $:$>3$1            now canonical form

# pass <route-addr>'s through
R<@$+>$*     $@<@$[$1$]>$2      resolve <route-addr>

# map colons to dots everywhere.....
R$*:$*       $1.$2           map colons to dots

# output local host in user@host.domain syntax
R$-       $1<@$w>            user w/o host
R$+<@$=w>    $:$1<@$w>       this host
R$+<@$->     $:$1<@$[$2$]>      canonicalize into dom
R$+<@$->     $:$1<@$2.$D>       if nameserver fails
R$+<@$=N.$D>    $@$1<@$2.$D>       nic-reg hosts are ok
R$+<@$*.$D>     $@$1%$2.$D<@$A>       else -> u%h@gateway

# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I>    $@$1<@$[$2.$3$]>   user@host.domain
R$+<@[$+]>      $@$1<@[$2]>        already ok

# Hide fake domains behind relays

R$+<@$+.BITNET>    $@$1%$2.BITNET<@$B>    user@host.BITNET
R$+<@$+.CSNET>     $@$1%$2.CSNET<@$C>     user@host.CSNET
R$+<@$+.UUCP>   $@$2!$1            user@host.UUCP




#####################
###   Rule Zero   ###
#####################


############################################################
############################################################
#####
#####     RULESET ZERO PREAMBLE
#####
##### The beginning of ruleset zero is constant through all
##### configurations.
```

```
#####
####################################################################
####################################################################

S0

# first make canonical
R$*<$*>$*       $1$2$3              defocus
R$+         $:$>3$1                 make canonical

# handle special cases
R$*<@[$+]>$*        $:$1<@$[[$2]$]>$3       numeric internet addr
R$*<@[$+]>$*        $#tcp$@[$2]$:$1@[$2]$3     numeric internet spec
R$+         $:$>6$1
R$-<@$w>        $#local$:$1
R@          $#error$:Invalid address   handle <> form

# canonicalize using the nameserver if not internal domain
R$*<@$*.$~I>$*        $:$1<@$[$2.$3$]>$4
R$*<@$->$*          $:$1<@$[$2$]>$3
R$*<@$->$*          $:$1<@$2.$D>$3          if nameserver fails

# now delete the local info
R<@$w>:$*       $@$>0$1             @here:... -> ...
R$*<@$w>        $@$>0$1             ...@here -> ...

###################################
#  End of ruleset zero preamble  #
###################################


###############################################
###   Machine dependent part of Rule Zero   ###
###############################################





# resolve fake top level domains by forwarding to other hosts
R$*<@$+.BITNET>$*   $#tcp$@$B$:$1<@$2.BITNET>$3      user@host.BITNET
R$*<@$+.CSNET>$* $#tcp$@$C$:$1<@$2.CSNET>$3    user@host.CSNET


# forward non-local UUCP traffic to our UUCP relay
R$*<@$*.UUCP>$*     $#tcpld$@$R$:$1<@$2.UUCP> uucp mail

# hide behind our internet relay when talking to people in the arpa domain
R$*<@$*.arpa>$*     $#tcp$@$2.arpa$:$1<@$2.arpa>$3    user@host.arpa
```

```
# but speak domains to them if they speak domains too
R$*<@$*>$*      $#tcpld$@$2$:$1<@$2>$3     user@host.domain

# remaining names must be local
R$+       $#local$:$1         everything else
```

E.3 UUCP Configuration File

The following file should be used on systems connected through UUCP only:

```
#
# Copyright (c) 1983 Eric P. Allman
# Copyright (c) 1988 The Regents of the University of California.
# All rights reserved.
#
# Redistribution and use in source and binary forms are permitted
# provided that the above copyright notice and this paragraph are
# duplicated in all such forms and that any documentation,
# advertising materials, and other materials related to such
# distribution and use acknowledge that the software was developed
# by the University of California, Berkeley.  The name of the
# University may not be used to endorse or promote products derived
# from this software without specific prior written permission.
# THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
# IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
# WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
#
#   @(#)uucpproto.mc 1.2 (Berkeley) 1/24/89
#   @(#)proto.mc 1.21 (Berkeley) 2/15/89
#
# built by keany on Fri Dec 22 08:23:24 PST 1989
# in /crg3/comm/keany/ptx_alpha/usr/src/bin/sendmail/cf/cf on crg3
#
############################################################
############################################################
#####
#####     SENDMAIL CONFIGURATION FILE
#####
############################################################
############################################################


##################
#   local info   #
##################

# file containing our internet aliases
```

```
Fw/usr/lib/sendmail.cw

# uucp hostnames
DUYOUR_UUCP_HOSTNAME_HERE
CUYOUR_UUCP_ALIASES_HERE

# local UUCP connections
CV  YOUR
CV  UUCP
CV  NEIGHBORS
CV  GO
CV  HERE


##############################
###   Setup Information   ###
##############################


######################
#   General Macros   #
######################

# local domain name
DDUUCP









# my official hostname
Dj$w




###############
#   Classes   #
###############

# Internal ("fake") domains that we use in rewriting
CIUUCP BITNET CSNET




######################
```

```
#   Version Number   #
######################

DZ1.35


######################
#   Special macros   #
######################

# my name
DnMAILER-DAEMON
# UNIX header format
DlFrom $g  $d
# delimiter (operator) characters
Do.:%@!^=/[]
# format of a total name
Dq$g$?x ($x)$.
# SMTP login message
De$j Sendmail $v/$Z ready at $b

###############
#   Options   #
###############

# location of alias file
OA/usr/lib/aliases
# wait up to ten minutes for alias file rebuild
Oa10
# substitution for space (blank) characters
OB.
# (don't) connect to "expensive" mailers
#Oc
# default delivery mode (deliver in background)
Odbackground
# temporary file mode
OF0600
# default GID
Og1
# location of help file
OH/usr/lib/sendmail.hf
# log level
OL9
# default network name
ONARPA
# default messages to old style
Oo
# queue directory
OQ/usr/spool/mqueue
# read timeout -- violates protocols
```

```
Or2h
# status file
OS/usr/lib/sendmail.st
# queue up everything before starting transmission
Os
# default timeout interval
OT3d
# time zone names (V6 only)
OtPST,PDT
# default UID
Ou1
# wizard's password
OW*
# load average at which we just queue messages
Ox8
# load average at which we refuse connections
OX12

###########################
#   Message precedences   #
###########################

Pfirst-class=0
Pspecial-delivery=100
Pbulk=-60
Pjunk=-100

#####################
#   Trusted users   #
#####################

Troot
Tdaemon
Tuucp

#########################
#   Format of headers   #
#########################

H?P?Return-Path: <$g>
HReceived: $?sfrom $s $.by $j ($v/$Z)
    id $i; $b
H?D?Resent-Date: $a
H?D?Date: $a
H?F?Resent-From: $q
H?F?From: $q
H?x?Full-Name: $x
HSubject:
# HPosted-Date: $a
# H?l?Received-Date: $b
```

```
H?M?Resent-Message-Id: <$t.$i@$j>
H?M?Message-Id: <$t.$i@$j>



###########################
###   Rewriting Rules   ###
###########################


################################
#  Sender Field Pre-rewriting  #
################################
S1
#R$*<$*>$*       $1$2$3          defocus

###################################
#  Recipient Field Pre-rewriting  #
###################################
S2
#R$*<$*>$*       $1$2$3          defocus



#################################
#  Final Output Post-rewriting  #
#################################
S4

R@        $@            handle <> error addr

# resolve numeric addresses to name if possible
R$*<@[$+]>$*     $:$1<@$[[$2]$]>$3      lookup numeric internet addr

# externalize local domain info
R$*<$+>$*     $1$2$3          defocus
R@$+:@$+:$+      @$1,@$2:$3          <route-addr> canonical

# UUCP must always be presented in old form
R$+@$-.UUCP      $2!$1           u@h.UUCP => h!u

# delete duplicate local names
R$+%$=w@$=w      $1@$w           u%host@host => u@host
R$+%$=w@$=w.$D      $1@$w            u%host@host => u@host


###########################
#  Name Canonicalization  #
###########################
S3
```

```
# handle "from:<>" special case
R$*<>$*         $@@           turn into magic token

# basic textual canonicalization -- note RFC733 heuristic here
R$*<$*<$*<$+>$*>$*>$*  $4           3-level <> nesting
R$*<$*<$+>$*>$*     $3           2-level <> nesting
R$*<$+>$*    $2           basic RFC821/822 parsing

# make sure <@a,@b,@c:user@d> syntax is easy to parse -- undone later
R@$+,$+          @$1:$2          change all "," to ":"

# localize and dispose of route-based addresses
R@$+:$+         $@$>6<@$1>:$2       handle <route-addr>

# more miscellaneous cleanup
R$+       $:$>8$1             host dependent cleanup
R$+:$*;@$+       $@$1:$2;@$3         list syntax
R$+:$*;         $@$1:$2;        list syntax
R$+@$+       $:$1<@$2>       focus on domain
R$+<$+@$+>       $1$2<@$3>       move gaze right
R$+<@$+>     $@$>6$1<@$2>       already canonical

# convert old-style addresses to a domain-based address
R$+^$+       $1!$2           convert ^ to !
R$-!$+       $@$>6$2<@$1.UUCP>      resolve uucp names
R$+.$-!$+    $@$>6$3<@$1.$2>        domain uucps
R$+!$+       $@$>6$2<@$1.UUCP>      uucp subdomains
R$+%$+       $:$>9$1%$2         user%host
R$+<@$+>     $@$>6$1<@$2>       already canonical
R$-.$+       $@$>6$2<@$1>       host.user


#################################
#   special local conversions   #
#################################

S6
R$*<@$=w>$*      $:$1<@$w>$3         get into u@$w form
R$*<@$=w.$D>$*      $:$1<@$w>$3
R$*<@$=U.UUCP>$* $:$1<@$w>$3

################################
#   Change rightmost % to @.   #
################################

S9
R$*%$*       $1@$2           First make them all @'s.
R$*@$*@$*    $1%$2@$3        Undo all but the last.
R$*@$*       $@$1<@$2>       Put back the brackets.
```

```
###################
###   Mailers   ###
###################


############################################################
############################################################
#####
#####     Local and Program Mailer specification
#####
############################################################
############################################################

Mlocal,   P=/bin/mail, F=rlsDFMmn, S=10, R=20, A=mail -d $u
Mprog, P=/bin/sh,   F=lsDFMe,   S=10, R=20, A=sh -c $u

S10
R@         $n        errors to mailer-daemon



############################################################
############################################################
#####
#####     Internet SMTP Mailer specification
#####
##### Messages processed by this specification are assumed to leave
##### the local domain -- hence, they must be canonical according to
##### RFC822 etc.  This means that machines not registered with
##### the NIC must be hidden behind our Internet relay.
#####
############################################################
############################################################

Mtcp,  P=[IPC], F=mDFMueXLC, S=14, R=24, A=IPC $h, E= r n

S14

# pass <route-addr>'s through
R<@$+>$*       $@<@$[$1$]>$2       resolve <route-addr>

# map colons to dots everywhere
R$*:$*         $1.$2           map colons to dots

# output local host in user@host.domain syntax
R$-       $1<@$w>              user w/o host
R$+<@$=w>      $:$1<@$w>       this host
R$+<@$->       $:$1<@$[$2$]>      canonicalize into dom
R$+<@$->       $:$1<@$2.$D>       if nameserver fails
R$+<@$=N.$D>      $@$1<@$2.$D>       nic-reg hosts are ok
```

```
R$+<@$*.$D>      $@$1%$2.$D<@$A>        else -> u%h@gateway

# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I>     $@$1<@$[$2.$3$]>    user@host.domain
R$+<@[$+]>       $@$1<@[$2]>         already ok

# output internal ("fake") domains as "user%host@relay"



R$+<@$+.UUCP>    $@$2!$1<@$w>        user@host.UUCP


S24

# put in <> kludge
R$*<$*>$*     $1$2$3            defocus
R$*       $:$>3$1               now canonical form

# pass <route-addr>'s through
R<@$+>$*      $@<@$[$1$]>$2        resolve <route-addr>

# map colons to dots everywhere.....
R$*:$*        $1.$2             map colons to dots

# output local host in user@host.domain syntax
R$-       $1<@$w>               user w/o host
R$+<@$=w>     $:$1<@$w>         this host
R$+<@$->      $:$1<@$[$2$]>         canonicalize into dom
R$+<@$->      $:$1<@$2.$D>          if nameserver fails
R$+<@$=N.$D>     $@$1<@$2.$D>          nic-reg hosts are ok
R$+<@$*.$D>      $@$1%$2.$D<@$A>          else -> u%h@gateway

# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I>     $@$1<@$[$2.$3$]>    user@host.domain
R$+<@[$+]>       $@$1<@[$2]>         already ok

# Hide fake domains behind relays



R$+<@$+.UUCP>    $@$2!$1             user@host.UUCP


##############################################################
##############################################################
#####
#####     UUCP Mailer specification
#####
```

```
###############################################################
###############################################################


Muucp, P=/usr/bin/uux, F=DFMhuU, S=13, R=23, M=100000,
   A=uux - -r -z -a$f -gC $h!rmail ($u)

S13
R$+        $:$>5$1             convert to old style
R$*<@$=w>$*       $1<@$w>$2        resolve abbreviations
R$*<@$->$*        $1<@$2.$D>$3        resolve abbreviations
R$+<@$+>      $2!$1            uucpize (no @'s in addr)
R$w!$+        $1            strip local name
R$+        $:$U!$1             stick on our host name
R$=U!$-%$-        $:$1!$2@$3.$D       ucbvax!user@host.domain

S23
R$+        $:$>5$1             convert to old style
R$*<@$=w>$*       $1<@$w>$2        resolve abbreviations
R$*<@$->$*        $1<@$2.$D>$3        resolve abbreviations
R$+<@$w>      $U!$1            a!b@here -> here!a!b
R$=U!$+           $2           here!a!b -> a!b
# sanity ... should not happen.
R$=U.$D!$+        $2           strip local name.domain


###############################################################
###############################################################
#####
#####     Provide Backward Compatibility
#####
###############################################################
###############################################################

########################################################
#  General code to convert back to old style names   #
########################################################
S5

R$+<@$w>      $1            strip host
R$+<@$-.UUCP>     $2!$1            u@host.UUCP => host!u



######################
###   Rule Zero    ###
######################


###############################################################
```

```
############################################################
#####
#####     RULESET ZERO PREAMBLE
#####
#####  The beginning of ruleset zero is constant through all
#####  configurations.
#####
############################################################
############################################################

S0

# first make canonical
R$*<$*>$*     $1$2$3           defocus
R$+      $:$>3$1               make canonical

# handle special cases
R$*<@[$+]>$*     $:$1<@$[[$2]$]>$3      numeric internet addr
R$*<@[$+]>$*     $#tcp$@[$2]$:$1@[$2]$3    numeric internet spec
R$+      $:$>6$1
R$-<@$w>      $#local$:$1
R@       $#error$:Invalid address  handle <> form

# canonicalize using the nameserver if not internal domain
R$*<@$*.$~I>$*      $:$1<@$[$2.$3$]>$4
R$*<@$->$*       $:$1<@$[$2$]>$3
R$*<@$->$*       $:$1<@$2.$D>$3         if nameserver fails

# now delete the local info
R<@$w>:$*     $@$>0$1             @here:... -> ...
R$*<@$w>      $@$>0$1             ...@here -> ...

##################################
#  End of ruleset zero preamble  #
##################################


###############################################
###   Machine dependent part of Rule Zero   ###
###############################################


# resolve local UUCP connections
R<@$=V.UUCP>:$+      $#uucp$@$1$:$2          @host.UUCP:...
R$+<@$=V.UUCP>       $#uucp$@$2$:$1          user@host.UUCP
```

```
# remaining names must be local
R$+        $#local$:$1          everything else
```

E.4 TCP/IP and UUCP Configuration Files

The following file should be used on a system that connects to a TCP/IP network and *also* connects to the outside world through UUCP:

```
#
# Copyright (c) 1983 Eric P. Allman
# Copyright (c) 1988 The Regents of the University of California.
# All rights reserved.
#
# Redistribution and use in source and binary forms are permitted
# provided that the above copyright notice and this paragraph are
# duplicated in all such forms and that any documentation,
# advertising materials, and other materials related to such
# distribution and use acknowledge that the software was developed
# by the University of California, Berkeley.  The name of the
# University may not be used to endorse or promote products derived
# from this software without specific prior written permission.
# THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
# IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
# WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
#
#   @(#)tcpuucpproto.mc 1.2 (Berkeley) 1/24/89
#   @(#)proto.mc 1.21 (Berkeley) 2/15/89
#
# built by keany on Fri Dec 22 08:23:26 PST 1989
# in /crg3/comm/keany/ptx_alpha/usr/src/bin/sendmail/cf/cf on crg3
#
############################################################
############################################################
#####
#####     SENDMAIL CONFIGURATION FILE
#####
############################################################
############################################################


##################
#   local info   #
##################

# file containing our internet aliases
Fw/usr/lib/sendmail.cw

# uucp hostnames
```

```
DUYOUR_UUCP_NAME_GOES_HERE
CUYOUR_UUCP_ALIASES_GO_HERE

# local UUCP connections
CV  YOUR
CV  UUCP
CV  NEIGHBORS
CV  GO
CV  HERE


##############################
###   Setup Information   ###
##############################


######################
#   General Macros   #
######################

# local domain name
DDYOUR_DOMAIN_GOES_HERE




# csnet relay host
DCYOUR_CSNET_RELAY_GOES_HERE

# bitnet relay host
DBYOUR_BITNET_RELAY_GOES_HERE

# my official hostname
Dj$w



###############
#   Classes   #
###############

# Internal ("fake") domains that we use in rewriting
CIUUCP BITNET CSNET



######################
#   Version Number   #
```

```
######################

DZ1.35


######################
#   Special macros   #
######################

# my name
DnMAILER-DAEMON
# UNIX header format
DlFrom $g  $d
# delimiter (operator) characters
Do.:%@!^=/[]
# format of a total name
Dq$g$?x ($x)$.
# SMTP login message
De$j Sendmail $v/$Z ready at $b

###############
#   Options   #
###############

# location of alias file
OA/usr/lib/aliases
# wait up to ten minutes for alias file rebuild
Oa10
# substitution for space (blank) characters
OB.
# (don't) connect to "expensive" mailers
#Oc
# default delivery mode (deliver in background)
Odbackground
# temporary file mode
OF0600
# default GID
Og1
# location of help file
OH/usr/lib/sendmail.hf
# log level
OL9
# default network name
ONARPA
# default messages to old style
Oo
# queue directory
OQ/usr/spool/mqueue
# read timeout -- violates protocols
Or2h
```

```
# status file
OS/usr/lib/sendmail.st
# queue up everything before starting transmission
Os
# default timeout interval
OT3d
# time zone names (V6 only)
OtPST,PDT
# default UID
Ou1
# wizard's password
OW*
# load average at which we just queue messages
Ox8
# load average at which we refuse connections
OX12

###########################
#   Message precedences   #
###########################

Pfirst-class=0
Pspecial-delivery=100
Pbulk=-60
Pjunk=-100

#####################
#   Trusted users   #
#####################

Troot
Tdaemon
Tuucp

#########################
#   Format of headers   #
#########################

H?P?Return-Path: <$g>
HReceived: $?sfrom $s $.by $j ($v/$Z)
   id $i; $b
H?D?Resent-Date: $a
H?D?Date: $a
H?F?Resent-From: $q
H?F?From: $q
H?x?Full-Name: $x
HSubject:
# HPosted-Date: $a
# H?l?Received-Date: $b
H?M?Resent-Message-Id: <$t.$i@$j>
```

```
H?M?Message-Id: <$t.$i@$j>


###########################
###   Rewriting Rules   ###
###########################


#################################
#  Sender Field Pre-rewriting  #
#################################
S1
#R$*<$*>$*        $1$2$3            defocus

####################################
#  Recipient Field Pre-rewriting  #
####################################
S2
#R$*<$*>$*        $1$2$3            defocus



#################################
#  Final Output Post-rewriting  #
#################################
S4

R@        $@            handle <> error addr

# resolve numeric addresses to name if possible
R$*<@[$+]>$*     $:$1<@$[[$2]$]>$3      lookup numeric internet addr

# externalize local domain info
R$*<$+>$*    $1$2$3          defocus
R@$+:@$+:$+      @$1,@$2:$3          <route-addr> canonical

# UUCP must always be presented in old form
R$+@$-.UUCP      $2!$1           u@h.UUCP => h!u

# delete duplicate local names
R$+%$=w@$=w      $1@$w           u%host@host => u@host
R$+%$=w@$=w.$D      $1@$w           u%host@host => u@host


###########################
#  Name Canonicalization  #
###########################
S3
```

```
# handle "from:<>" special case
R$*<>$*            $@@            turn into magic token

# basic textual canonicalization -- note RFC733 heuristic here
R$*<$*<$*<$+>$*>$*>$*  $4            3-level <> nesting
R$*<$*<$+>$*>$*     $3           2-level <> nesting
R$*<$+>$*    $2            basic RFC821/822 parsing

# make sure <@a,@b,@c:user@d> syntax is easy to parse -- undone later
R@$+,$+            @$1:$2           change all "," to ":"

# localize and dispose of route-based addresses
R@$+:$+            $@$>6<@$1>:$2       handle <route-addr>

# more miscellaneous cleanup
R$+       $:$>8$1               host dependent cleanup
R$+:$*;@$+        $@$1:$2;@$3          list syntax
R$+:$*;           $@$1:$2;         list syntax
R$+@$+        $:$1<@$2>        focus on domain
R$+<$+@$+>        $1$2<@$3>        move gaze right
R$+<@$+>      $@$>6$1<@$2>        already canonical

# convert old-style addresses to a domain-based address
R$+^$+        $1!$2            convert ^ to !
R$-!$+        $@$>6$2<@$1.UUCP>      resolve uucp names
R$+.$-!$+     $@$>6$3<@$1.$2>        domain uucps
R$+!$+        $@$>6$2<@$1.UUCP>      uucp subdomains
R$+%$+        $:$>9$1%$2          user%host
R$+<@$+>      $@$>6$1<@$2>        already canonical
R$-.$+        $@$>6$2<@$1>        host.user


#################################
#   special local conversions   #
#################################

S6
R$*<@$=w>$*      $:$1<@$w>$3        get into u@$w form
R$*<@$=w.$D>$*      $:$1<@$w>$3
R$*<@$=U.UUCP>$* $:$1<@$w>$3

################################
#   Change rightmost % to @.   #
################################

S9
R$*%$*        $1@$2          First make them all @'s.
R$*@$*@$*     $1%$2@$3       Undo all but the last.
R$*@$*        $@$1<@$2>      Put back the brackets.
```

```
###################
###   Mailers   ###
###################


############################################################
############################################################
#####
#####     Local and Program Mailer specification
#####
############################################################
############################################################

Mlocal,   P=/bin/mail, F=rlsDFMmn, S=10, R=20, A=mail -d $u
Mprog, P=/bin/sh,   F=lsDFMe,     S=10, R=20, A=sh -c $u

S10
R@        $n        errors to mailer-daemon


############################################################
############################################################
#####
#####     Local Domain SMTP Mailer specification
#####
##### Messages processed by this specification are assumed to remain
##### the local domain.  Hence, they can refer to hosts that are
##### not registered in the NIC host table.
#####
############################################################
############################################################

Mtcpld,   P=[IPC], F=mDFMueXLC, S=17, R=27, A=IPC $h, E= r n

S17

# cleanup forwarding a bit
R$*<$*>$*     $1$2$3          defocus
R$*       $:$>3$1             canonicalize
R$*%$*<@$w>      $:$>9$1%$2         user%localhost@localdomain

# pass <route-addr>'s through
R<@$+>$*      $@<@$[$1$]>$2       resolve <route-addr>

# map colons to dots everywhere
R$*:$*        $1.$2           map colons to dots

# output local host as user@host.domain
R$-       $@$1<@$w>       user w/o host
R$+<@$w>      $@$1<@$w>       this host
```

```
R$+<@$=w>    $@$1<@$w>        or an alias
R$+<@$->     $:$1<@$[$2$]>       ask nameserver
R$+<@$w>     $@$1<@$w>        this host
R$+<@$->     $@$1<@$2.$D>        if nameserver fails

# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I>     $@$1<@$[$2.$3$]>    user@host.domain
R$+<@[$+]>       $@$1<@[$2]>         already ok

# output fake domains as user%fake@relay

R$+<@$+.BITNET>     $@$1%$2.BITNET<@$B>    user@host.bitnet
R$+<@$+.CSNET>      $@$1%$2.CSNET<@$C>     user@host.CSNET
R$+<@$+.UUCP>    $@$2!$1<@$w>        user@host.UUCP


S27

# cleanup
R$*<$*>$*     $1$2$3           defocus
R$*       $:$>3$1              now canonical form
R$*%$*<@$w>      $:$>9$1%$2          user%localhost@localdomain

# pass <route-addr>'s through
R<@$+>$*      $@<@$[$1$]>$2       resolve <route-addr>

# map colons to dots everywhere
R$*:$*        $1.$2            map colons to dots

# output local host as user@host.domain
R$-       $@$1<@$w>        user w/o host
R$+<@$w>      $@$1<@$w>        this host
R$+<@$=w>     $@$1<@$w>        or an alias
R$+<@$->      $:$1<@$[$2$]>       ask nameserver
R$+<@$w>      $@$1<@$w>        this host
R$+<@$->      $@$1<@$2.$D>        if nameserver fails

# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I>     $@$1<@$[$2.$3$]>    user@host.domain
R$+<@[$+]>       $@$1<@[$2]>         already ok

# output fake domains as user%fake@relay

R$+<@$+.BITNET>     $@$1%$2.BITNET<@$B>    user@host.BITNET
R$+<@$+.CSNET>      $@$1%$2.CSNET<@$C>     user@host.CSNET
R$+<@$+.UUCP>    $@$2!$1             user@host.UUCP


############################################################
```

```
############################################################
#####
#####     Internet SMTP Mailer specification
#####
#####  Messages processed by this specification are assumed to leave
#####  the local domain -- hence, they must be canonical according to
#####  RFC822 etc.  This means that machines not registered with
#####  the NIC must be hidden behind our Internet relay.
#####
############################################################
############################################################

Mtcp,  P=[IPC], F=mDFMueXLC, S=14, R=24, A=IPC $h, E= r n

S14

# pass <route-addr>'s through
R<@$+>$*     $@<@$[$1$]>$2       resolve <route-addr>

# map colons to dots everywhere
R$*:$*       $1.$2            map colons to dots

# output local host in user@host.domain syntax
R$-       $1<@$w>             user w/o host
R$+<@$=w>    $:$1<@$w>        this host
R$+<@$->     $:$1<@$[$2$]>       canonicalize into dom
R$+<@$->     $:$1<@$2.$D>        if nameserver fails
R$+<@$=N.$D>    $@$1<@$2.$D>        nic-reg hosts are ok
R$+<@$*.$D>     $@$1%$2.$D<@$A>        else -> u%h@gateway

# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I>    $@$1<@$[$2.$3$]>    user@host.domain
R$+<@[$+]>      $@$1<@[$2]>         already ok

# output internal ("fake") domains as "user%host@relay"

R$+<@$+.BITNET>    $@$1%$2.BITNET<@$B>    user@host.BITNET
R$+<@$+.CSNET>     $@$1%$2.CSNET<@$C>     user@host.CSNET
R$+<@$+.UUCP>   $@$2!$1<@$w>        user@host.UUCP


S24

# put in <> kludge
R$*<$*>$*    $1$2$3           defocus
R$*       $:$>3$1             now canonical form

# pass <route-addr>'s through
R<@$+>$*     $@<@$[$1$]>$2       resolve <route-addr>
```

```
# map colons to dots everywhere.....
R$*:$*       $1.$2            map colons to dots

# output local host in user@host.domain syntax
R$-       $1<@$w>             user w/o host
R$+<@$=w>    $:$1<@$w>        this host
R$+<@$->     $:$1<@$[$2$]>       canonicalize into dom
R$+<@$->     $:$1<@$2.$D>        if nameserver fails
R$+<@$=N.$D>     $@$1<@$2.$D>        nic-reg hosts are ok
R$+<@$*.$D>      $@$1%$2.$D<@$A>        else -> u%h@gateway

# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I>     $@$1<@$[$2.$3$]>    user@host.domain
R$+<@[$+]>       $@$1<@[$2]>         already ok

# Hide fake domains behind relays

R$+<@$+.BITNET>     $@$1%$2.BITNET<@$B>    user@host.BITNET
R$+<@$+.CSNET>      $@$1%$2.CSNET<@$C>     user@host.CSNET
R$+<@$+.UUCP>    $@$2!$1             user@host.UUCP



############################################################
############################################################
#####
#####     UUCP Mailer specification
#####
############################################################
############################################################


Muucp, P=/usr/bin/uux, F=DFMhuU, S=13, R=23, M=100000,
   A=uux - -r -z -a$f -gC $h!rmail ($u)

S13
R$+       $:$>5$1             convert to old style
R$*<@$=w>$*      $1<@$w>$2        resolve abbreviations
R$*<@$->$*       $1<@$2.$D>$3        resolve abbreviations
R$+<@$+>     $2!$1            uucpize (no @'s in addr)
R$w!$+       $1            strip local name
R$+       $:$U!$1             stick on our host name
R$=U!$-%$-       $:$1!$2@$3.$D       ucbvax!user@host.domain

S23
R$+       $:$>5$1             convert to old style
R$*<@$=w>$*      $1<@$w>$2        resolve abbreviations
R$*<@$->$*       $1<@$2.$D>$3        resolve abbreviations
R$+<@$w>     $U!$1            a!b@here -> here!a!b
R$=U!$+          $2           here!a!b -> a!b
```

```
# sanity ... should not happen.
R$=U.$D!$+       $2           strip local name.domain


############################################################
############################################################
#####
#####     Provide Backward Compatibility
#####
############################################################
############################################################

#####################################################
#  General code to convert back to old style names  #
#####################################################
S5

R$+<@$w>      $1           strip host
R$+<@$-.UUCP>    $2!$1           u@host.UUCP => host!u



#####################
###   Rule Zero   ###
#####################


############################################################
############################################################
#####
#####     RULESET ZERO PREAMBLE
#####
#####  The beginning of ruleset zero is constant through all
#####  configurations.
#####
############################################################
############################################################

S0

# first make canonical
R$*<$*>$*    $1$2$3           defocus
R$+       $:$>3$1           make canonical

# handle special cases
R$*<@[$+]>$*     $:$1<@$[[$2]$]>$3      numeric internet addr
R$*<@[$+]>$*     $#tcp$@[$2]$:$1@[$2]$3    numeric internet spec
R$+       $:$>6$1
R$-<@$w>     $#local$:$1
R@        $#error$:Invalid address  handle <> form
```

```
# canonicalize using the nameserver if not internal domain
R$*<@$*.$~I>$*      $:$1<@$[$2.$3$]>$4
R$*<@$->$*       $:$1<@$[$2$]>$3
R$*<@$->$*       $:$1<@$2.$D>$3          if nameserver fails

# now delete the local info
R<@$w>:$*     $@$>0$1            @here:... -> ...
R$*<@$w>      $@$>0$1            ...@here -> ...

##################################
#  End of ruleset zero preamble  #
##################################


###############################################
###   Machine dependent part of Rule Zero   ###
###############################################



# resolve local UUCP connections
R<@$=V.UUCP>:$+     $#uucp$@$1$:$2          @host.UUCP:...
R$+<@$=V.UUCP>      $#uucp$@$2$:$1          user@host.UUCP


# resolve fake top level domains by forwarding to other hosts
R$*<@$+.BITNET>$*   $#tcp$@$B$:$1<@$2.BITNET>$3      user@host.BITNET
R$*<@$+.CSNET>$* $#tcp$@$C$:$1<@$2.CSNET>$3    user@host.CSNET



# resolve SMTP traffic
R$*<@$+>$*       $#tcpld$@$2$:$1<@$2>$3 user@host.domain

# remaining names must be local
R$+       $#local$:$1          everything else
```

F. Removing TCP/IP

F. Removing TCP/IP

Appendix F
Removing the Installed ptx/TCP/IP Software

F.1 Introduction F-1

F.2 Preparing to Remove the Software F-1

F.3 Removing the Software F-1

Appendix F
Removing the Installed ptx/TCP/IP Software

F.1 Introduction

This appendix explains the proper procedure for removing the ptx/TCP/IP software.

F.2 Preparing to Remove the Software

Before removing the ptx/TCP/IP software, be sure to let the users know that you will be removing the software and the ptx/TCP/IP services will not be available. You can use the DYNIX/ptx **wall** command to broadcast a message to all users on your system.

F.3 Removing the Software

You can use the ptx/ADMIN menu system to remove the ptx/TCP/IP software. Note that you must have root privileges to execute selections from the ptx/ADMIN menu.

The following steps describe how to remove the ptx/TCP/IP software. For more information about the DYNIX/ptx commands in this procedure, refer to your DYNIX/ptx operating system documentation.

1. Log in as **root** or use the **su** command to become the superuser.
2. After using the **wall** command to notify the users, bring the system down to single-user mode. If you're not familiar with the correct procedure for bringing the system down to single-user mode, refer to your DYNIX/ptx operating system documentation.
3. While in single-user mode, use the **menu** command to invoke the ptx/ADMIN menu system.
4. When the first menu screen appears, select "System Administration." When the System Administration menu appears, select "Software Management." When the Software Management menu appears, select "Remove Software Package." When the menu system prompts you for the name of the software package to remove, enter *tcp*. Then follow the instructions from the menu system to remove the TCP/IP software.

G. *Tuning Kernel* Parameters

G. Tuning Kernel Parameters

Appendix G

Tuning the ptx/TCP/IP Kernel Parameters

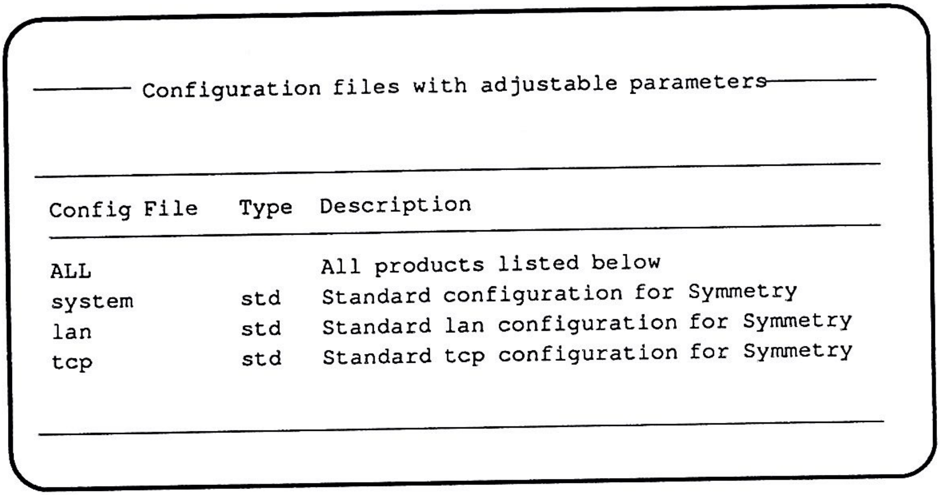
ARPTAB_BSIZ G-4
ARPTAB_NB G-4
DL_MAX_NETS G-4
IPFORWARDING G-4
IPSENDREDIRE G-4
MAX_IP_PROTO G-4
MAX_REASSQ G-4
MAX_SUBNETS G-4
N_TCP_PCB_FR G-4
N_TCP_PCB_HD G-5
N_UDP_PCB_FR G-5
N_UDP_PCB_HD G-5
ROUTETAB_BSI G-5
ROUTETAB_SIZ G-5
UDPCHKSUM G-5

Appendix G

Tuning the ptx/TCP/IP Kernel Parameters

This appendix describes the ptx/TCP/IP kernel parameters that can be tuned using the ptx/ADMIN menu system. After installing the ptx/TCP/IP software, you can access the tcp `Tunable Parameters` menu window through the following sequence:

1. Type **menu** at the superuser prompt (#) to enter the ptx/ADMIN menu system.
2. Type A on the clmain menu window to select the `System Administration` menu window.
3. Type C on the clmgmt menu window to select the `Kernel Configuration` menu window.
4. Type k on the `Compile or Configure a Kernel` menu window.
5. Type A at the `Visibility level for parameter changes` entry on the `kconfig` menu window and press Run (usually F1 or Ctrl-F) to display the following `Configuration files with adjustable parameters` menu window:



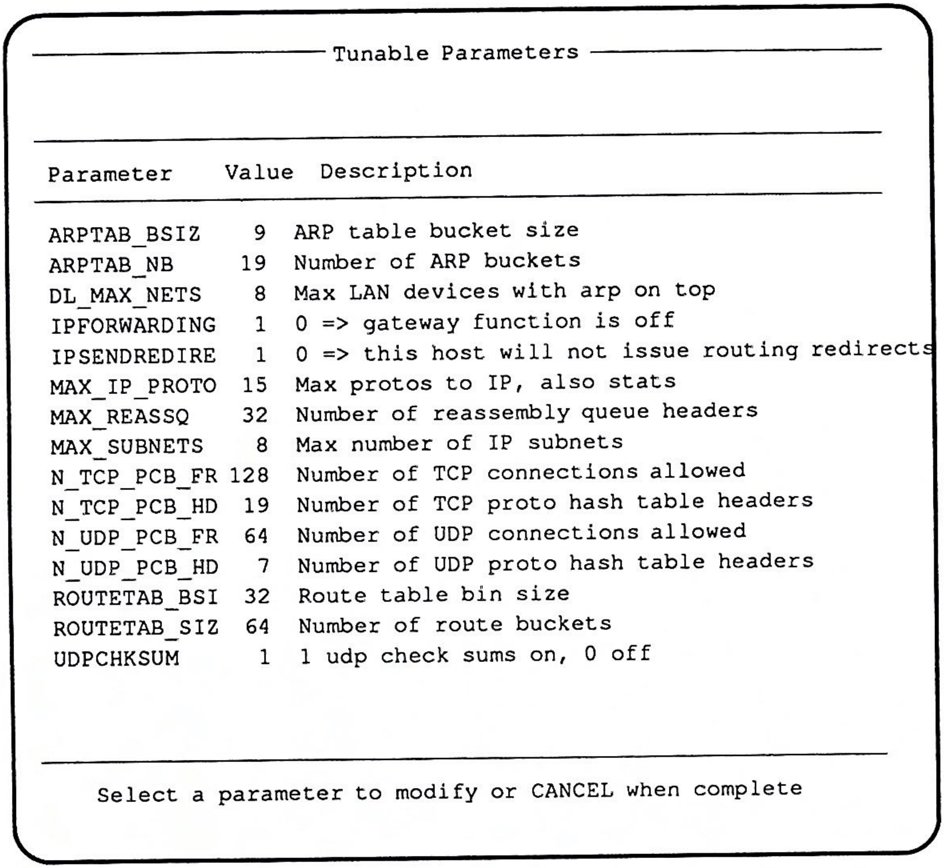
```
——————— Configuration files with adjustable parameters———————

──────────────────────────────────────────────────────────────

 Config File    Type  Description
 ─────────────────────────────────────────────────────────────
 ALL                  All products listed below
 system         std   Standard configuration for Symmetry
 lan            std   Standard lan configuration for Symmetry
 tcp            std   Standard tcp configuration for Symmetry

──────────────────────────────────────────────────────────────
```

Move the menu bar to the system configuration file containing the tcp parameters then press Run (usually F1 or Ctrl-F) to display the following tcp `Tunable Parameters` menu window:



```
                    Tunable Parameters

 Parameter     Value  Description

 ARPTAB_BSIZ     9   ARP table bucket size
 ARPTAB_NB      19   Number of ARP buckets
 DL_MAX_NETS     8   Max LAN devices with arp on top
 IPFORWARDING    1   0 => gateway function is off
 IPSENDREDIRE    1   0 => this host will not issue routing redirects
 MAX_IP_PROTO   15   Max protos to IP, also stats
 MAX_REASSQ     32   Number of reassembly queue headers
 MAX_SUBNETS     8   Max number of IP subnets
 N_TCP_PCB_FR  128   Number of TCP connections allowed
 N_TCP_PCB_HD   19   Number of TCP proto hash table headers
 N_UDP_PCB_FR   64   Number of UDP connections allowed
 N_UDP_PCB_HD    7   Number of UDP proto hash table headers
 ROUTETAB_BSI   32   Route table bin size
 ROUTETAB_SIZ   64   Number of route buckets
 UDPCHKSUM       1   1 udp check sums on, 0 off

     Select a parameter to modify or CANCEL when complete
```

Move the menu bar to the parameter you want to view or change, then press Run (usually F1 or Ctrl-F).

When you modify a parameter, keep in mind that other parameters might be dependent on the value of the parameter you modify. A change to one parameter can change the value of other parameters.

The parameters shown in the following sections can be configured through the menu system.

ARPTAB_BSIZ

Number of entries in an arp table hashing "bucket".

ARPTAB_NB

Number of arp table "buckets" allocated in the table.

DL_MAX_NETS

Number of subnetworks that the ARP module will support.

IPFORWARDING

Turn IP packet forwarding on or off (on by default) set to 0 to disable IP packet forwarding.

IPSENDREDIRE

Allow the sending of IP level redirects for network level routing and error information. On by default.

MAX_IP_PROTO

The maximum number of upper level protocols that IP will support.

MAX_REASSQ

Number of IP reassembly headers allocated. Used for the reassembly of large IP level datagrams. Use netstat -m to monitor the consumption of this resource. Lost network packets may cause this parameter to look artificially small. Running out of reassembly headers has the biggest impact on NFS performance.

MAX_SUBNETS

The maximum number of network interfaces supported by IP.

N_TCP_PCB_FR

Number of protocol control blocks that TCP has allocated this. Protocol control blocks are used to maintain the connection state information for a given TCP connection. In addition PCB's are used for incoming connection indications. Set this number to the maximum number of TCP connections your system will require with some additional for network servers and system overhead. Monitor the number in use with the **netstat -m** command. Allocation failures will be indicated. Related parameters are the number of TLI modules (NTLI) and the number of TIDWR modules (NTIRDWR). When increasing the number of TCP endpoints be aware that the number of streams data buffers probably will need to be adjusted also. Specifically each tcp connection requires a 64 byte streams buffer for a TCP header template. TCP outgoing data utilizes 128 byte buffers for the TCP/IP header in addition to any user data.

N_TCP_PCB_HD

Number of hash table headers used for locating TCP connections. Should be a prime number of best lookup performance.

N_UDP_PCB_FR

This parameter configures the number of UDP endpoints that are required.

N_UDP_PCB_HD

Number of hash table headers used for locating UDP endpoints. Should be a prime number for best lookup performance.

ROUTETAB_BSI

Number of routing hash headers allocated in IP. Systems connected to large number of networks may wish to tune this parameter to improve routing lookup performance.

ROUTETAB_SIZ

Number of routing entries in IP. Systems with large numbers of routes should use **netstat -m** to monitor the number of free routing entries and adjust this parameter accordingly.

UDPCHKSUM

Turn outgoing UDP packet checksumming on or off by default. Currently on by default. It is not recommended to turn this parameter off.

H. Kernel Error Messages

Appendix H
Kernel Error Messages

H.1 Introduction H-1
H.2 WARNING Messages H-2
H.3 NOTE Messages H-6

Appendix H
Kernel Error Messages

H.1 Introduction

This appendix contains ptx/TCP/IP kernel error messages. The messages are grouped by type: WARNING and NOTE. When you need to look up an error message, go to the section of this appendix that corresponds to the message type.

In this appendix, when error messages contain variable data such as a drive number, that portion of the message is specified by the corresponding **printf()** conversion characters. The **printf** string begins with `%`. If the next character is `x`, the actual error message will contain a hexadecimal value in that position. If the next character is `d`, the actual error message will contain a signed decimal value in that position. If the next character is `s`, the actual error message will contain a character string in that position. Refer to **printf**(3S) for more information about these characters.

Error messages starting with a **printf** string are located at the beginning of each section.

H.2 WARNING Messages

WARNING messages print on the console when the kernel encounters a recoverable error condition. These errors do not affect the entire system. Typically, they are caused by situations such as an operation attempting to allocate an entry in a data structure that is already full, or a hardware device getting an error that causes an operation to fail.

Most of the following error messages suggest a corrective action. Typically, if a data structure is filled, you can reconfigure the kernel to add more space. If a filesystem is full, you can clean up the filesystem or add additional disks. If you have a hardware problem that you cannot correct, contact your service representative.

WARNING: arpresolve: no free entry

No free entries were available in the arp table (too many hosts). Increase the number of arp table slots and hash buckets.

WARNING: ip%d: failure to add dev to devsw0

The ip multiplexor failed to add an entry to the device switch table.

WARNING: ip: failure to make clone device

The IP multiplexor could not add a clone device entry.

WARNING: ip_lrsrv: unknown M_CTL message 0x%x0

An unknown IP control code was processed by the IP lower read service routine.

WARNING: ip_lrsrv: unknown message type0x%x0

An unknown streams message type was received by the IP lower read service routine.

WARNING: ip_ioctl: I_LINK -- no free subnet entries0

The IP multiplexor could not allocate a structure to connect to a network device. Increase the MAX_SUBNETS parameter.

WARNING: icmp_error: type 0x%x too large

An attempt was made to send an invalid ICMP control code. The ICMP packet is discarded.

WARNING: ADD IP OUTPUT sum for proto 0x%x0

A protocol error has occurred from a layer above IP. This error is a result of an invalid upper protocol or a buffer corruption problem.

WARNING: rtfree: called with NULL entry

An invalid structure was passed to the rtfree routine.

WARNING: rt_ioctl: Bad control code 0x%x0

An invalid ioctl control code was passed to the IP routing driver.

WARNING: loop%d: failure to add dev to devsw0

The IP loopback device could not be added to the device switch table.

WARNING: loopopen: CLONEOPEN ???0

An attempt was made to call the IP loopback device as a clone device.

WARNING: loopopen: minor dev not 00

The minor device code passed to the IP loopback open routine was not 0.

WARNING: tcp%d: failure to add dev to devsw0

The TCP multiplexor could not be added to the device switch tables.

WARNING: tcpmux: failure to make clone device

A clone entry could not be added for the TCP multiplexor.

WARNING: tcpmux: out of free PCB structures0

The TCP multiplexor was unable to allocate a protocol structure. Increase the number of available TCP endpoints through the N_PCB_TCP_FREE parameter. Also adjust the number of free TLI (NTLI) modules, and the number of free tirdwr (NTRW) modules.

WARNING: tcpmux_ursrv: (0x%x), NULL PCB0

The TCP upper read service routine was called on on a queue that did not have a valid protocol control block.

WARNING: tcpmux_uwsrv(0x%x): NULL PCB0

The TCP upper read service routine was run on a queue that did not have a valid protocol control block entry.

WARNING: udp%d: failure to add dev to devsw0

A device switch table entry could not be added for the UDP streams multiplexor.

WARNING: udp: failure to make clone device

A clone device entry could not be added for the UDP streams multiplexor.

WARNING: pts: TCGETA data block allocb failed0

An attempt to allocate a streams data buffer for TCGETA parameters failed.

WARNING: pts: TCGETA received data block bad size0

A TCGETA ioctl with a preallocated buffer does not have enough allocated space to store the requested information.

WARNING: pts: TCGETP data block allocb failed0

An attempt to allocate a streams data buffer for TCGETP parameters failed.

WARNING: rlmod: TCGETP received data block bad size0

A TCGETP ioctl with a preallocated buffer does not have enough allocated space to store the requested information.

WARNING: rlmod: TCSETAF FLUSHR flush msg failed0

The driver failed an attempt to send a flush message upstream after processing a TCSETAF message.

WARNING: tnmod: TCGETA data block allocb failed0

An attempt to allocate a streams data buffer for TCGETA parameters failed.

WARNING: tnmod: TCGETA received data block bad size0

A TCGETA ioctl with a preallocated buffer does not have enough allocated space to store the requested information.

WARNING: tnmod: TCGETP data block allocb failed0

An attempt to allocate a streams data buffer for TCGETP parameters failed.

WARNING: tnmod: TCGETP received data block bad size0

A TCGETP ioctl with a preallocated buffer does not have enough allocated space to store the requested information.

WARNING: tnmod: TCSETAF FLUSHR flush msg failed0

The driver failed an attempt to send a flush message upstream after processing a TCSETAF message.

WARNING: tnmod: TN_TERM data block allocb failed0

An attempt to allocate a streams data buffer for TN_TERM parameters failed.

H.3 NOTE Messages

NOTE messages are not really errors. They do not affect system operations; the system continues to run normally after the message appears.

These messages occur when something happens that was not expected by the kernel. The operation being executed might fail and return an error to the user who invoked the operation. The NOTE message appears on the console to alert the administrator to the condition.

These messages are sometimes due to hardware or software configuration problems. When you see one of these messages on the console, be sure to check your system configuration.

NOTE: tcp_input: t_maxseg <= 0 sent from (0x%x)0

The machine with the given IP addresses attempted to negotiate a TCP maximum segment size of less than 0, in violation of the TCP protocol specification. This value will be ignored.

Index

Index

Index

.rhosts 3-9

/bin/mail 5-1

/bin/rmail 5-1

/etc/ftpusers 3-9

/etc/hosts 2-10; 4-1

/etc/hosts.equiv 3-8

/etc/inetd.conf 3-4

/etc/init.d/netservers 4-7

/etc/init.d/tcp 2-9

/etc/named 4-7

/etc/netconf 2-2

/etc/networks 3-1

/etc/protocols 3-2

/etc/services 3-3

*/usr/hosts/** 3-6

/usr/lib/libinet.a 4-1

A

Adjusting kernel parameters 1-10

Administration overview 1-1
  address resolution 1-4
  gateways 1-3
  mail 1-5
  name service 1-4
  protocols 1-3
  routing 1-3

Administrative concepts 1-1

B

Base network files 2-1,2
  /etc/hosts 2-1,10
  /etc/init.d/tcp 2-1,9
  /etc/netconf 2-1,2

Berkeley Internet Name Domain 4-1

BIND 4-1
  named 4-1

  resolver 4-1

C

Caching only server 4-2
Caching server 4-2
Client-server model description 1-3
Configuration flowchart 1-11
Configuration prerequisites 1-9
  DYNIX/ptx version 1-9
  hardware 1-9
  host name selection 1-9
  worksheets completed 1-9

D-E

Domains 4-1
Enabling rwhod 3-10
  manually 3-10
  rebooting 3-10

F-G-H

Gateways 1-3
gethostbyaddr 4-1
gethostbyname 4-1

I

IN-ADDR.ARPA domain 4-6
Installation overview 1-9
Internet address selection 1-6
Internet v. internet vii

J-K-L-M

kernel error messages
  NOTE messages H-6
  WARNING messages H-2
Mail 5-1
Manual organization vii
Master servers 4-2

N

Name server file types
  boot 4-3
  domain data 4-3
  Table 4-1 4-3
Name server preparation 4-6
Name server types 4-2
  caching 4-2
  caching only server 4-2
  master servers 4-2
  primary master server 4-2
  remote server 4-2
  secondary master server 4-2
Named configuration worksheet 4-7
Named files
  named.boot 4-4
  named.ca 4-5
  named.hosts 4-5
  named.hosts.rev 4-6
  named.local 4-5
  resolv.conf 4-4
named.boot 4-4
named.ca 4-5
named.hosts 4-5
named.hosts.rev 4-6
named.local 4-5
Netstat 3-11
Network administration 3-1
Network checks 2-12
Network classes 1-7
  class A 1-7
  class B 1-7
  class C 1-7
Network Information Center 4-1
Network Information Center (NIC) 1-8
Network information files 3-1
  /etc/inetd.conf 3-4
  /etc/networks 3-1
  /etc/protocols 3-2
  /etc/services 3-3
  */usr/hosts/** 3-6

Network permissions files 3-6
  .rhosts 3-9
  /etc/ftpusers 3-9
  /etc/hosts.equiv 3-8
Network permissions flowchart (Figure 3-1) 3-7
Network utilities 3-10
  arp 3-10
  netstat 3-11
  ping 3-13
  route 3-14
NIC 4-1

O-P

Ping 3-13
Primary master server 4-2
Protocols
  GGP 1-3
  ICMP 1-3
  IP 1-3
  TCP 1-3
  UDP 1-3
ptx/TCP/IP processes 1-5

Q

Quick configuration 1-13
  /etc/hosts 1-14
  /etc/netconf 1-13

R

Related publications viii
Remote server 4-2
resolv.conf 4-4
Route 3-14
Route utility 1-4
Routing 1-3

S

Sample named configuration 4-7
  /etc/named.boot 4-8
  /etc/named.hosts 4-9
  /etc/named.hosts.rev 4-9
  /etc/named.local 4-8
  /etc/resolv.conf 4-8
Sample network (Figure 1-1) 1-1
Sample network (Figure 2-1) 2-2
Saving files 1-9
Secondary master server 4-2
Sending 8-bit mail D-13
Sendmail 5-1
  phases 5-2
  Sendmail structure (Figure 5-1) 5-1
  sendmail.cf 5-2
Sendmail installation guide D-1
Sendmail setup 5-3
  creating newaliases 5-3
  starting 5-3
  testing 5-3
sendmail.cf 5-2
standard Sendmail files
  tcpproto.cf 5-2
  tcpuucpproto.cf 5-2
  uucpproto.cf 5-2
Starting named 4-7
  /etc/init.d/netservers 4-7
  /etc/named 4-7
Starting ptx/TCP/IP 2-11
Subdomains 4-1

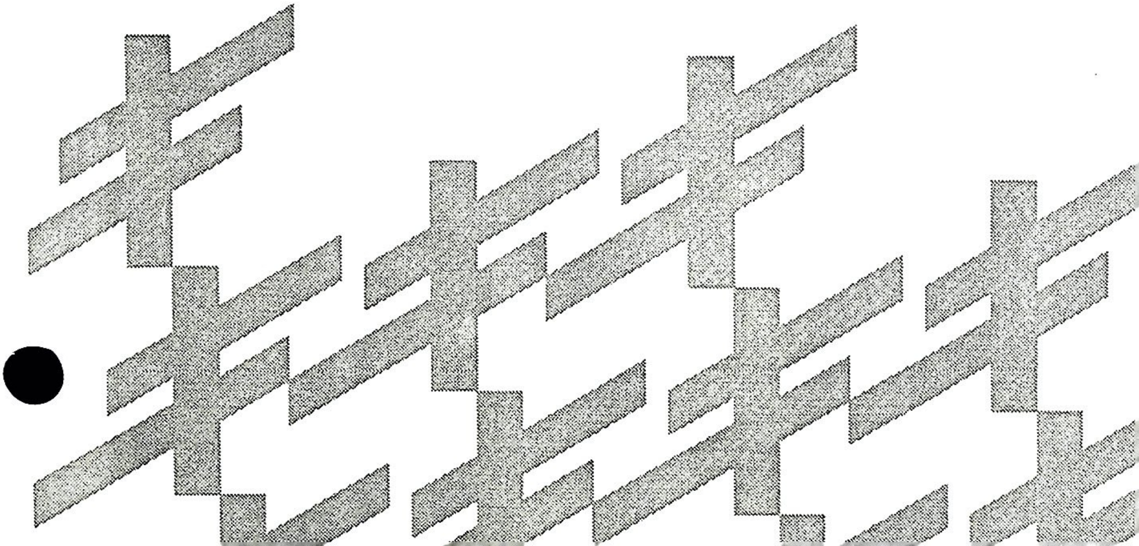
T-U-V-W-X-Y-Z

tcpproto.cf 5-3
tcpuucpproto.cf 5-3
UNIX mail routing 5-1
User experience vii
UUCP 5-1
uucpproto.cf 5-3
Zone of authority 4-5

ptx/TCP/IP Programming Manual

ptx/TCP/IP™ Programming Manual

1003-48919-00
Sequent Computer Systems, Inc.



Symmetry is a registered trademark of Sequent Computer Systems, Inc.

DYNIX/ptx and ptx/NET are trademarks of Sequent Computer Systems, Inc.

Ethernet is a trademark of Xerox Corp.

UNIX is a trademark of AT&T.

Copyright © 1990 by Sequent Computer Systems, Inc. All rights reserved. This document may not be copied or reproduced in any form without permission from Sequent Computer Systems, Inc. Information in this document is subject to change without notice.

Printed in the United States of America.

Contents

About This Manual

Assumptions About the Reader v
Organization v
Related Publications vi

Chapter 1. The Transport Interface

1.1 Introduction 1-1
1.2 Service Modes 1-2
1.3 Handling Errors 1-3

Chapter 2. Programmer Access to Network Databases

2.1 Introduction 2-1
2.2 System Header Files 2-2
2.3 Hostname Routines 2-2
2.4 Network Name Routines 2-5
2.5 Service Name Routines 2-7
2.6 Endpoint Address Routines 2-9
2.7 Network Byte Order Conversion Routines 2-11
2.8 Internet Address Manipulation Routines 2-13

Chapter 3. Transmission Control Protocol (TCP)

3.1 Introduction 3-1
3.2 Local Management Routines 3-1
3.3 Connection Establishment Routines 3-14
3.4 Data Transfer Routines 3-18
3.5 Connection Release Routines 3-19

Chapter 4. User Datagram Protocol (UDP)

4.1 Introduction 4-1
4.2 Local Management Routines 4-1
4.3 Data Transfer Routines 4-3

Chapter 5. Porting from Sockets to the TLI

5.1 Introduction 5-1
5.2 Handling Signals 5-1
5.3 Porting a TCP Server 5-2
5.4 Porting a TCP Client 5-5
5.5 Using Read and Write Calls in a TLI Application 5-7
5.6 Porting a UDP Program 5-8
5.7 Porting Examples 5-9

Appendix A. Programming Examples

A.1 Introduction A-1
A.2 TCP Programming Examples A-1
A.3 UDP Programming Examples A-14

Figures

1-1 The Transport Level Interface 1-1

Tables

5-1 Socket and TLI Routines for a TCP Server 5-2
5-2 Socket and TLI Routines for a TCP Client 5-6
5-3 Socket and TLI Routines for a UDP Program 5-8

About This Manual

This manual describes the Transport Interface services available to the ptx/TCP/IP programmer. The Transport Interface, commonly referred to as the Transport Layer Interface (TLI), was introduced in AT&T's UNIX System V Release 3. The TLI provides the basic interface to the underlying STREAMS modules that support both the connection-oriented Transmission Control Protocol (TCP) and connectionless-oriented User Datagram Protocol (UDP). STREAMS is a facility that provides a standard interface for character input and output.

This manual also describes how to access network databases and how to port an application from the Berkeley sockets programming interface to the TLI.

Assumptions About the Reader

This manual assumes the reader is familiar with the information in the *DYNIX/ptx Network Programmer's Guide*. The guide provides a general description of the TLI routines and their use. This manual also assumes the reader has a knowledge of the C programming language and the STREAMS facility associated with the operating system. Refer to the "Related Publications," section if you need additional programming information.

Organization

This manual is organized as follows:

- Chapter 1, "The Transport Interface," describes programming fundamentals and the TLI.
- Chapter 2, "Programmer Access to Network Databases," describes the various routines used to access the network databases.
- Chapter 3, "Transmission Control Protocol (TCP)," describes the local management, connection establishment, data transfer, and connection release routines associated with the Transmission Control Protocol (TCP).
- Chapter 4, "User Datagram Protocol (UDP)," describes the local management and data transfer routines associated with the User Datagram Protocol (UDP).

- Chapter 5, "Porting From Sockets to the TLI," describes how to port an application using the BSD sockets facility to the TLI facility.
- Appendix A, "Programming Examples," provides examples of both TCP and UDP programs.

Related Publications

The documents listed below provide additional information that may be helpful to the ptx/TCP/IP programmer:

- *ptx/TCP/IP and Applications Overview*, Sequent Computer Systems, 1990, part number 1003-48916
- *STREAMS Overview*, Sequent Computer Systems, 1990, part number 1003-48615
- *STREAMS Application Programming Guide*, Sequent Computer Systems, 1990, part number 1003-48616
- *DYNIX/ptx Network Programmer's Guide*, Sequent Computer Systems, 1990, part number 1003-49229
- *The C Programming Language*, Brian W. Kernighan, Dennis Ritchie, Prentice-Hall Software Series, 1978

1. The Transport Interface

1. The Transport Interface

Chapter 1

The Transport Interface

1.1 Introduction 1-1

1.2 Service Modes 1-2
  Connection Mode 1-2
  Connectionless Mode 1-2

1.3 Handling Errors 1-3

Figures

1-1 The Transport Level Interface 1-1

Chapter 1
The Transport Interface

1.1 Introduction

The Transport Level Interface (TLI) allows communications applications (transport users) to access the services of a transport provider. As shown in Figure 1-1, applications reside in user space while the underlying transport provider is implemented as a STREAMS module. The TLI acts as an interface in that it specifies the transport services available to the applications. Most importantly, the combination of the TLI and transport provider insulate an application from the underlying network protocols. A lower network protocol can be changed or a new protocol installed with minimal effect on user applications.

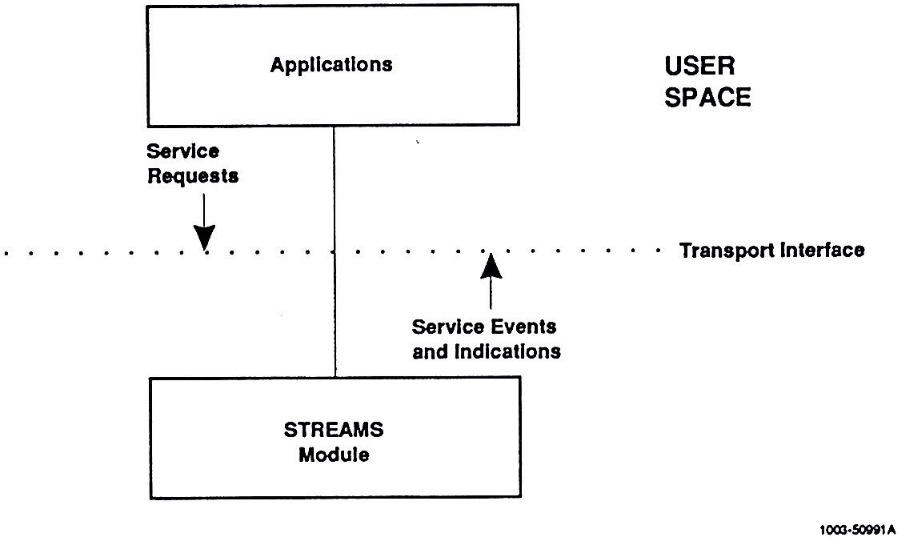

Applications
USER SPACE
Service Requests
Transport Interface
Service Events and Indications
STREAMS Module
1003-50991A

Figure 1-1. The Transport Level Interface. The Transport Level Interface allows applications to access the underlying transport provider.

The TLI was modeled after the industry standard Transport Service Definition (1 8072). It is sometimes referred to as the Transport Library Interface as it is implemented through library routines on top of the STREAMS mechanism. Many services available to STREAMS applications are also available to users of the TLI.

1.2 Service Modes

The TLI provides two modes of service, connection mode and connection-less mode. The transmission control protocol (TCP) specifies a connection-mode service. The user datagram protocol (UDP) specifies a connectionless-mode service. The following sections describe both service modes in detail.

1.2.1 Connection Mode

Connection-mode service is circuit-oriented, and enables data to be transferred reliably, in sequence, over an established connection. It also provides an identification mechanism that avoids the overhead of address resolution and transmission during the data transfer phase. This service is attractive for applications that require relatively long-lived, circuit-oriented interactions.

A connection-mode transport service consists of four phases:

- Local management
- Connection establishment
- Data transfer
- Connection release

1.2.2 Connectionless Mode

Connectionless mode, or as it sometimes called, *datagram* service, is message-oriented. It supports data transfer in self-contained units with no logical relationship required among multiple units. Each unit of data transmitted is entirely self-contained. This service requires only a preexisting association between the peer users involved, which determines the characteristics of the data to be transmitted. All the information required to deliver a unit of data (for example, the destination address) is presented to the transport provider, together with the data to be transmitted, in one service access. One service access need not relate to any other service access.

Connectionless-mode service is attractive for applications that meet the following criteria:

- Involve short-term request/response interactions
- Exhibit a high level of redundancy
- Are dynamically reconfigurable
- Do not require guaranteed, in-sequence delivery of data

A connectionless-mode transport service consists of two phases, local management and data transfer.

1.3 Handling Errors

All TLI routines return 0 upon successful completion and –1 on failure. When a routine fails, the variable **t_errno** is set to indicate the type of error. Use the **t_error** routine to print a message that describes the error. If **t_errno** is set to TSYSERR (which indicates a system error occurred), the variable **errno** is set to the type of system error. In this case, **t_error** prints messages describing the values in **t_errno** and **errno**.

For information about the errors each routine can generate, see the TLI man pages.

2. Programmer Access to Network Databases

2. Programmer Access to Network Databases

Chapter 2

Programmer Access to Network Databases

2.1 Introduction 2-1

2.2 System Header Files 2-2

2.3 Host Name Routines 2-2
gethostbyname 2-3
gethostbyaddr 2-3
sethostent 2-4
endhostent 2-4

2.4 Network Name Routines 2-5
getnetbyname 2-5
getnetbyaddr 2-6
setnetent 2-6
getnetent 2-6
endnetent 2-6

2.5 Service Name Routines 2-7
getservbyname 2-7
getservbyport 2-8
setservent 2-9
getservent 2-9
endservent 2-9

2.6 Endpoint Address Routines 2-9
getmyinaddr 2-9
getpeerinaddr 2-10

2.7 Network Byte Order Conversion Routines 2-11
htonl 2-11
htons 2-11
ntohl 2-12
ntohs 2-12

2.8 Internet Address Manipulation Routines 2-13
inet_addr 2-13
inet_network 2-13
inet_makeaddr 2-13

inet_lnaof 2-14
inet_netof 2-14
inet_ntoa 2-14

Chapter 2
Programmer Access to Network Databases

2.1 Introduction

At times, it is necessary to locate and construct network addresses when using the interprocess communication facilities in ptx/TCP/IP. ptx/TCP/IP provides library routines that allow you to determine network addresses. Most of the routines retrieve address information from files known as network databases. These network databases contain the names of hosts and network services, protocol port numbers, and other related information. In addition, two other routines retrieve the addresses of the local and remote endpoints associated with a connection.

The network database routines retrieve information in several ways. Some routines search a local database file and return the first entry that matches a specified network identifier (such as a host name). Other routines simply return the next entry in the database file. The `gethostbyname` and `gethostbyaddr` routines can also use a service called the Berkeley Internet Name Domain (BIND) Server. If BIND is active on the local system, these routines can retrieve information from databases maintained on remote systems.

The network database routines provide a way to map one type of network identifier to another. For example, if you know the name of a host, you can use the `gethostbyname` routine to obtain the host's network address. Conversely, if you know a host's address but not its name, you can use the `gethostbyaddr` to derive the host's name.

The library routines let you obtain information about the following network entities:

- Hosts
- Networks
- Network Services
- Endpoint addresses

2.2 System Header Files

When writing applications to use ptx/TCP/IP network database routines, you need to include one or more of the following system header files:

- The file <*sys/types.h*> is always required.
- The file <*netdb.h*> defines the functions and structures used to map names of hosts, networks, and services into numeric values.
- The file <*netinet/in.h*> defines the constants and structures used when specifying Internet addresses and making control requests on TCP and UDP endpoints.

2.3 Host Name Routines

The host name routines retrieve information about hosts on the network. If BIND is not active on your system, the routines retrieve entries from the */etc/hosts* database file. If BIND is active, the `gethostbyname` and `gethostbyaddr` routines use the name server rather than accessing the */etc/hosts* file. Because the name server has no concept of the next line in a database file, the `gethostent` routine cannot be used while the name server is active.

Each entry represents a mapping from host name to host address. The routines return a pointer to a *hostent* structure that contains information from one entry.

The *hostent* structure contains the following members:

```
struct  hostent {
        char   *h_name;        /* official name of host */
        char   **h_aliases;    /* alias list            */
        int    h_addrtype;     /* host address type     */
        int    h_length;       /* length of address     */
        char   **h_addr_list;  /* list of addresses from
                                  name server */
#define  h_addr  h_addr_list[0]  /* first address,
                                    for compatibility */
};
```

h_name is the official name of the host. **h_aliases** is a NULL terminated array of alternate names for the host. **h_addrtype** is the type of address being returned; currently this is always AF_INET. **h_length** is the length, in bytes, of the address. **h_addr_list** is a pointer to a list of network addresses for the host. The identifier **h_addr** is defined as the first address in the list for compatibility with previous versions of the *hostent* structure. Host addresses are returned as long integers and in network standard byte order.

2.3.1 gethostbyname

The `gethostbyname` routine takes a host name and returns a *hostent* structure. The routine has the following format:

```
struct hostent *gethostbyname(name)
char *name;
```

The `gethostbyname` routine determines if BIND is active. If so, it uses the name server to retrieve a *hostent* structure for the specified host name. If BIND is not active, the routine searches from the beginning of the */etc/hosts* database file and returns the first entry that matches the specified host name. If no matching entry is found or an error occurs, the routine returns a null pointer (0). Each `gethostbyname` call that accesses */etc/hosts* opens and closes the file, unless a previous `sethostent` call specified the file should not be closed.

2.3.2 gethostbyaddr

The `gethostbyaddr` routine takes a host address and returns a *hostent* structure. The routine has the following format:

```
struct hostent *gethostbyaddr(addr, len, type)
char *addr;
int len, type;
```

The `gethostbyaddr` routine determines if BIND is active. If so, it uses the name server to retrieve a *hostent* structure for the specified address. If BIND is not active, the routine searches from the beginning of the */etc/hosts* database file and returns the first entry that matches the specified address. If no matching entry is found or an error occurs, the routine returns a null pointer (0). Each `gethostbyaddr` call that accesses */etc/hosts* opens and closes the file, unless a previous `sethostent` call specified the file should not be closed.

2.3.3 sethostent

The `sethostent` routine opens the */etc/hosts* database file and sets the pointer at the beginning of the file. The routine has the following format:

```
sethostent(stayopen)
int stayopen;
```

If the **stayopen** flag is nonzero, the database file is not closed after each call to `gethostbyname` or `gethostbyaddr`. For applications that access the file many times, keeping it open avoids the overhead of repeatedly opening and closing the file.

2.3.4 endhostent

The `endhostent` routine closes the */etc/hosts* database file. The routine has the following format:

```
endhostent()
```

2.4 Network Name Routines

The network name routines retrieve entries from the */etc/networks* database file. Each entry, or line, in the database file represents a network name to network number mapping. The routines return a pointer to a *netent* structure that contains information from one entry in the file.

The *netent* structure contains the following members:

```
struct   netent  {
     char     *n_name;       /* official name of network */
     char     **n_aliases;   /* alias list               */
     int      n_nettype;     /* network number type      */
     ulong    n_net;         /* network number           */
};
```

n_name is the official name of the network. **n_aliases** is a zero terminated list of alternate names for the network. **n_nettype** is the type of the network number returned; currently this is always AF_INET. **n_net** is the network number. Network numbers are returned in local host byte order.

2.4.1 getnetbyname

The `getnetbyname` routine takes a network name and returns a *netent* structure. The routine has the following format:

```
struct netent *getnetbyname(name)
char *name;
```

The `getnetbyname` routine sequentially searches from the beginning of the */etc/networks* database file and returns the first entry that matches the specified network name. If no matching entry is found or an error occurs, the routine returns a null pointer (0). Each `getnetbyname` call opens and closes the database file, unless a previous `setnetent` call specified the file should not be closed.

2.4.2 getnetbyaddr

The `getnetbyaddr` routine takes a network number and returns a *netent* structure. The routine has the following format:

```
struct netent *getnetbyaddr(net)
int net;
```

The `getnetbyaddr` routine sequentially searches from the beginning of the */etc/networks* database file and returns the first entry that matches the specified network number. If no matching entry is found or an error occurs, the routine returns a null pointer (0). Each `getnetbyaddr` call opens and closes the database file, unless a previous `setnetent` call specified the file should not be closed.

2.4.3 setnetent

The `setnetent` routine opens the */etc/networks* database file and sets the pointer at the beginning of the file. The routine has the following format:

```
setnetent(stayopen)
int stayopen;
```

If the **stayopen** flag is nonzero, the database file is not closed after each call to `getnetbyname`, `getnetbyaddr`, or `getnetent`.

2.4.4 getnetent

The `getnetent` routine retrieves the next entry in the */etc/networks* database file. The routine has the following format:

```
struct netent *getnetent()
```

Each `getnetent` call opens and closes the database file, unless a previous `setnetent` call specified the file should not be closed.

2.4.5 endnetent

The `endnetent` routine closes the */etc/networks* database file. The routine has the following format:

```
endnetent()
```

2.5 Service Name Routines

The service name routines can be used to find the port number for a common service (such as FTP) or the service associated with a known port. The routines retrieve entries from the */etc/services* database file. Each entry, or line, in the database file represents a service name, port number, and protocol mapping. The routines return a pointer to a *servent* structure that contains information from one entry in the file.

The *servent* structure contains the following members:

```
struct  servent {
        char      *s_name;       /* official service name */
        char      **s_aliases;   /* alias list            */
        int       s_port;        /* port number           */
        char      *s_proto;      /* protocol to use       */
};
```

s_name is the official name of the service. **s_aliases** is a zero terminated list of alternate names for the service. **s_port** is the port number at which the service resides. Port numbers are returned in network-standard byte order. **s_proto** is the name of the protocol to use when contacting the service.

A service is expected to reside at a specific port and employ a particular communication protocol. However, a service may reside on multiple ports or support multiple protocols. If either of these occurs, you may need to write a routine that alternately retrieves (using `getservent`) and tests database entries until the desired port and protocol combination is found.

2.5.1 getservbyname

The `getservbyname` routine takes a service name and, optionally, a qualifying protocol, and returns a *servent* structure. The routine has the following format:

```
struct servent *getservbyname(name, proto)
char *name, *proto;
```

The `getservbyname` routine sequentially searches from the beginning of the */etc/services* database file and returns the first entry that matches the specified search criteria. The search criteria may be either a service name by itself or a service name and protocol combination.

For example, the following call returns the service specification for a TELNET server using any protocol.

```
sp = getservbyname("telnet", (char *)0);
```

The follwoing call returns only that TELNET server which uses the TCP protocol.

```
sp = getservbyname("telnet", "tcp");
```

If no matching entry is found or an error occurs, the routine returns a null pointer (0). Each `getservbyname` call opens and closes the database file, unless a previous `setservent` call specified the file should not be closed.

2.5.2 getservbyport

The `getservbyport` routine takes a port number and, optionally, a qualifying protocol, and returns a *servent* structure. The routine has the following format:

```
struct servent *getservbyport(port, proto)
int port;
char *proto;
```

The `getservbyport` routine sequentially searches from the beginning of the */etc/services* database file and returns the first entry that matches the specified search criteria. The search criteria may be either a port number by itself or a port number and protocol combination.

For example, the following call returns the service specification on port 43 using any protocol.

```
sp = getservbyport(43, (char *)0);
```

The following call returns the service specification on port 43 that uses the TCP protocol.

```
sp = getservbyport(43, "tcp");
```

If no matching entry is found or an error occurs, the routine returns a null pointer (0). Each `getservbyport` call opens and closes the database file, unless a previous `setservent` call specified the file should not be closed.

2.5.3 setservent

The `setservent` routine opens the */etc/services* database file and sets the pointer at the beginning of the file. The routine has the following format:

```
setservent(stayopen)
int stayopen;
```

If the **stayopen** flag is nonzero, the database file is not closed after each call to `getservbyname`, `getservbyport`, or `getservent`.

2.5.4 getservent

The `getservent` routine retrieves the next entry in the */etc/services* database file. The routine has the following format:

```
struct servent *getservent()
```

Each `getservent` call opens and closes the database file, unless a previous `setservent` call specified the file should not be closed.

2.5.5 endservent

The `endservent` routine closes the */etc/services* database file. The routine has the following format:

```
endservent()
```

2.6 Endpoint Address Routines

The endpoint address routines return the addresses of the local and remote endpoints associated with a connection. These routines are useful in applications that do not explicitly specify an Internet address when binding an endpoint.

2.6.1 getmyinaddr

The `getmyinaddr` routine returns the Internet and port address for a local endpoint. The routine has the following format:

```
getmyinaddr(fd, sin, sinlen)
int fd;
struct sockaddr_in *sin;
int sinlen;
```

fd must refer to a local open TCP or UDP endpoint. **sin** must be a buffer large enough to hold an address in the **struct sockaddr_in** format. **sinlen** must be initialized to **sizeof(struct sockaddr_in)**.

2.6.2 getpeerinaddr

The `getpeerinaddr` routine returns the Internet and port address for a remote endpoint. The routine has the following format:

```
getpeerinaddr(fd, sin, sinlen)
int fd;
struct sockaddr_in *sin;
int sinlen;
```

fd must refer to a local open TCP or UDP endpoint. **sin** must be a buffer large enough to hold an address in the **struct sockaddr_in** format. **sinlen** must be initialized to **sizeof(struct sockaddr_in)**.

2.7 Network Byte Order Conversion Routines

Internet defines a standard byte order for integers. However, the internal representation of integers (their byte order) differs from machine to machine. Thus, the byte order of integers on a particular machine may not be the byte order that Internet expects. For example, the byte order used by the i386 processors in the Symmetry Series is the reverse of the Internet byte order.

ptx/TCP/IP provides four library routines that convert between local host byte order and network-standard byte order. All four routines take a value as an argument and return a new value with the bytes rearranged. When writing original TCP/IP programs or porting existing code, use the appropriate conversion routine to copy a short or long value between the local machine and the network.

The routines differ in the type of conversion they perform and the size of the integer they convert. For example, the `ntohs` routine converts a short (2-byte) integer from network-standard byte order to local host byte order. Conversely, the `htons` routine converts a short integer from local host byte order to network-standard byte order. Two other routines (`ntohl` and `htonl`) perform similar conversions on long (4-byte) integers.

2.7.1 htonl

The `htonl` routine converts a long (4-byte) integer to network-standard byte order. The routine has the following format:

```
netorder = htonl(hostorder);
ulong netorder, hostorder;
```

2.7.2 htons

The `htons` routine converts a short (2-byte) integer in the host's local byte order to a 2-byte integer in network-standard byte order. The routine has the following format:

```
netorder = htons(hostorder);
short int netorder, hostorder;
```

2.7.3 ntohl

The `ntonl` routine converts a long (4-byte) integer in network-standard byte order to a 4-byte integer in local host byte order. The routine has the following format:

```
hostorder = ntohl(netorder);
ulong netorder, hostorder;
```

2.7.4 ntohs

The `ntons` routine converts a short (2-byte) integer in network byte order to a 2-byte integer in the local host byte order. The routine has the following format:

```
hostorder = ntohs(netorder);
short int netorder, hostorder;
```

2.8 Internet Address Manipulation Routines

Internet addresses are often shown in a format known as *dot notation*. In this notation, Internet addresses are written as decimal integers separated by decimal points, where each integer represents one byte in the address. Because some programs need to translate between 32-bit Internet addresses and dot notation, ptx/TCP/IP provides three routines that perform the translation. Other routines let you construct Internet addresses and break apart the addresses into network numbers and local addresses.

In the following routines, Internet addresses are returned in network standard byte order, while network numbers and local addresses are returned in local host byte order.

2.8.1 inet_addr

The `inet_addr` routine takes an address expressed using dot notation and returns the corresponding 32-bit Internet address. The routine has the following format:

```
ulong inet_addr(cp)
char *cp;
```

2.8.2 inet_network

The `inet_network` routine takes an address expressed using dot notation and returns the corresponding Internet network number. The routine has the following format:

```
ulong inet_network(cp)
char *cp;
```

2.8.3 inet_makeaddr

The `inet_makeaddr` routine combines an Internet network address with the local address of a host on that network. The routine has the following format:

```
struct in_addr inet_makeaddr(net, lna)
int net, lna;
```

2.8.4 inet_lnaof

The `inet_lnaof` routine takes an Internet address and extracts the local network address. The routine has the following format:

```
int inet_lnaof(in)
struct in_addr in;
```

2.8.5 inet_netof

The `inet_netof` routine takes an Internet address and extracts the network number. The routine has the following format:

```
int inet_netof(in)
struct in_addr in;
```

2.8.6 inet_ntoa

The `inet_ntoa` routine performs the inverse of **inet_addr** by translating a 32-bit Internet address into an ASCII string in Internet dot format. The routine has the following format:

```
char *inet_ntoa(in)
struct in_addr in;
```

3. Transmission Control Protocol

3. Transmission
Control Protocol

Chapter 3

Transmission Control Protocol (TCP)

3.1 Introduction 3-1

3.2 Local Management Routines 3-1
t_alloc 3-2
t_bind 3-3
t_close 3-4
t_error 3-5
t_free 3-5
t_getinfo 3-6
t_getstate 3-7
t_look 3-7
t_open 3-8
t_optmgmt 3-9
TCP_SETOPT 3-12
t_sync 3-13
t_unbind 3-14

3.3 Connection-Establishment Routines 3-14
t_accept 3-14
t_connect 3-15
t_listen 3-16
t_rcvconnect 3-17

3.4 Data Transfer Routines 3-18
t_rcv 3-18
t_snd 3-19

3.5 Connection Release Routines 3-19
t_rcvdis 3-19
t_rcvrel 3-20
t_snddis 3-20
t_sndrel 3-21

Chapter 3

Transmission Control Protocol (TCP)

3.1 Introduction

This chapter describes the TLI routines that an application (transport user) uses to access the TCP driver (transport provider). TCP provides a connection-mode, end-to-end communication service. The following phases characterize the TCP service:

- Local management
- Connection establishment
- Data transfer
- Connection release

All TCP applications must include the <*tiuser.h*> and <*tihdr.h*> header files. The *nsl* and *inet* libraries must be specified at build time. Any TLI routines using the **sockaddr_in** address format require the <*netinet/in.h*> header file. Routines using the **tcp_options** structure require the <*netinet/netinet.h*> header.

3.2 Local Management Routines

The local management routines support operations between an application and the TCP driver. Most importantly, these routines open a transport endpoint and bind an address to the endpoint.

The TCP local management phase uses the following routines:

- **t_alloc** – Allocate memory for a TLI structure.
- **t_bind** – Bind a transport endpoint to a local Internet address.
- **t_close** – Close a transport endpoint.
- **t_error** – Print a TLI error message.

- **t_free** – Free memory allocated for a TLI structure with the **t_alloc** routine.
- **t_getinfo** – Retrieve information about a transport endpoint.
- **t_getstate** – Get the state of the transport endpoint.
- **t_look** – Return the current event on a transport endpoint.
- **t_open** – Open a transport endpoint.
- **t_optmgmt** – Negotiate options with the TCP driver.
- **t_sync** – Synchronize a transport endpoint with the TCP driver.
- **t_unbind** – Place a transport endpoint in the unbound state.

3.2.1 t_alloc

The **t_alloc** routine allocates memory for the various transport structures. The structures are used as arguments to the TLI routines. Using **t_alloc** to allocate TLI structures helps ensure that your programs will be compatible with future releases of the transport interface. The routine has the following format:

```
char *t_alloc(fd, struct_type, fields)
int fd;
int struct_type;
int fields;
```

fd is the transport endpoint to which the created structure applies. **struct_type** specifies the type of structure to allocate and can be any of the following values:

- T_BIND
- T_CALL
- T_OPTMGMT
- T_DIS
- T_UNITDATA
- T_UDERROR

- T_INFO

All of the above structures, except T_INFO, contain at least one field of the type **netbuf**. The **fields** argument specifies whether a buffer should also be allocated for fields of this type. Specify **fields** as the bitwise-OR of any of the following:

- T_ADDR – The **addr** field of the **t_bind**, **t_call**, **t_unitdata**, or **t_uderr** structures.
- T_OPT – The **opt** field of the **t_optmgmt**, **t_call**, **t_unitdata**, or **t_uderr** structures.
- T_UDATA – The **udata** field of the **t_call**, **t_discon**, or **t_unitdata** structures.
- T_ALL – All relevant fields of the given structure.

NOTE

Care must be taken when using pointers to buffers allocated by **t_alloc**. *These pointers are used by* **t_free** *and* **t_close** *(which calls* **t_free***) when returning memory used by the buffers to free memory space. If a pointer refers to some other area of memory, rather than the original buffer, this free memory space may be corrupted.*

3.2.2 t_bind

The **t_bind** routine binds an address to a given transport endpoint opened with the **t_open** routine. This routine has the following format:

```
int t_bind(fd, req, ret)
int fd;
struct t_bind *req;
struct t_bind *ret;
```

req and **ret** refer to a **t_bind** structure that contains the following members:

```
struct t_bind {
   struct netbuf addr; /* buffer containing
                          a struct sockaddr_in */
   unsigned qlen;      /* listen or connect    */
}
```

If the **req** structure is null or the **sockaddr_in** address is INADDR_ANY, TCP creates a unique port identifier and binds the local IP address to the interface used for the connect request (if **qlen** equals 0). If the port address is within the reserved range specified by IPPORT_RESERVED, then the endpoint **t_open** must have been done by a process running as **root** (**uid** equals 0).

In most cases, if a port address is specified and it is already in use, an error is returned. However, special cases do exist. For details, see the description of the TP_REUSEADDR option in the **t_optmgmt** routine.

The **qlen** parameter specifies how many connection indications can be pending at one time. Setting **qlen** to 1 is generally acceptable for servers that expect few connect indications. If **qlen** is greater than 1, the server must be able to process multiple connect indications.

If **qlen** is greater than the maximum allowed by the TCP driver, the **ret** structure will contain the maximum value and will use it. In this case, the port address must be specified as part of the **req** structure. On machines that have several Internet addresses, the IP address should usually be set to INADDR_ANY so that all requests can be honored. In some cases, however, an application may only want to listen on certain Internet addresses. For more information on the **qlen** parameter, see the description of the **t_accept** routine.

3.2.3 t_close

The **t_close** routine closes a transport endpoint specified by **fd** without exiting. This routine has the following format:

```
int t_close(fd)
int fd;
```

Calling **t_close** on a transport connection that has already been closed causes the **t_close** routine to return with all resources freed. Calling **t_close** on a closing or active transport connection with a nonzero linger time causes an orderly release. In this case, the **t_close** routine returns when the last queued data sent on the transport connection is transmitted or when the linger time expires. With a zero linger time, the connection is abortively closed and any queued data is discarded.

3.2.4 t_error

The **t_error** routine sends a TLI error message to the standard error output. The message describes the last error encountered during a call to a TLI routine. The **t_error** routine has the following format:

```
void t_error(errmsg)
char *errmsg;
```

errmsg is a user-supplied string that gives context to the error. **t_errno** contains a value that represents the last error. **t_errlist** is an array of message strings that describe TLI errors and **t_nerr** is the maximum index into the array. The **t_error** routine uses **t_errno** as an index into the array to obtain the related message string.

In some cases, additional TCP error information can be found in the **errno** variable. Use the **perror** library routine to print the value of **errno**.

3.2.5 t_free

The **t_free** routine frees memory previously allocated with the **t_alloc** routine.

NOTE

Care must be taken when using pointers to buffers allocated by **t_alloc**. *These pointers are used by* **t_free** *and* **t_close** *(which calls* **t_free**) *when returning memory used by the buffers to free memory space. If a pointer refers to some other area of memory, rather than the original buffer, this free memory space may be corrupted.*

The **t_free** routine has the following format:

```
int t_free(ptr, struct_type)
char *ptr;
int struct_type;
```

The **ptr** field points to a structure that was previously allocated by **t_alloc**.

The **struct_type** field indicates the type of structure pointed to by **ptr**. **struct_type** can be any one of the following values:

- T_BIND
- T_CALL
- T_OPTMGMT
- T_DISCON
- T_UNITDATA
- T_UDERROR
- T_INFO

3.2.6 t_getinfo

The **t_getinfo** routine returns the parameters associated with the **t_info** structure. This routine has the following format:

```
int t_getinfo(fd, info)
int fd;
struct t_info *info;
```

The **t_info** structure contains the following members:

```
struct t_info {
   long addr;        /* sizeof(struct sockaddr_in) */
   long options;     /* sizeof(struct tcp_options) */
   long tsdu;        /* 0, not supported by TCP    */
   long etsdu;       /* -1, no limit               */
   long connect;     /* -2, no data on connect     */
   long discon;      /* -2, no data on disconnect  */
   long servtype;    /* T_COTS or T_COTS_ORD       */
}
```

3.2.7 t_getstate

The **t_getstate** routine returns the current state of the TCP driver associated with the transport endpoint specified by **fd**. The format of the routine is as follows:

```
int t_getstate(fd)
int fd;
```

The current state may be one of the following:

- T_DATAXFER – The TCP driver has established a connection and can transfer data.
- T_IDLE – The TCP endpoint is bound.
- T_INCON – An incoming connection is pending.
- T_INREL – The TCP endpoint is waiting for an orderly release request.
- T_OUTCON – An outgoing connection is pending.
- T_OUTREL – The TCP endpoint is waiting for an orderly release indication.
- T_UNBND – The TCP endpoint was just opened and is in the unbound state.

3.2.8 t_look

The **t_look** routine returns a value that indicates the current event on a transport endpoint. The routine has the following format:

```
int t_look(fd)
int fd;
```

fd is the transport endpoint that the routine checks for an event.

This routine allows the TCP driver to notify a TCP user of an asynchronous event. The routine returns one of the following events:

- T_CONNECT – A connect confirmation was received.
- T_DATA – Normal data was received.
- T_DISCONNECT – A disconnect was received.

- T_ERROR – A fatal error occurred.
- T_EXDATA – Expedited was received.
- T_LISTEN – A connection indication was received.
- T_ORDREL – An orderly release indication was received.
- T_UDERR – A datagram error occurred.

3.2.9 t_open

The **t_open** routine can be used to open an endpoint for a TCP connection. This routine should be called with the following arguments:

```
int t_open(path, oflag, info)
char *path;
int oflag;
struct t_info *info;
```

path should be set to TLI_TCP (defined in <*netinet/in.h*>).

oflag should be set to O_RDWR, which allows reading and writing through the endpoint. **oflag** may also be set to O_NDELAY, which places the endpoint in nonblocking mode.

An optional **info** parameter may also be specified. **info** points to a **t_info** structure that returns TCP information. On a successful return, **info** contains the following fields:

- **addr** – Contains the maximum size (in bytes) of a transport protocol address.
- **options** – Contains the size (in bytes) of the **tcp_options** structure.
- **tsdu** – Contains the maximum size of a *transport service data unit (tsdu).* TCP is a byte stream and, therefore, it has no tsdu.
- **etsdu** – Contains the maximum size of an *expedited transport service data unit (etsdu).* In TCP, there is no limit on this size, so this field is always set to –1.
- **connect** – Contains the maximum amount of data allowed to be passed to the **t_connect** routine TCP does not allow data to be sent during connection establishment, therefore, **long connect** is set to –2.

- **discon** – Contains the maximum amount of data allowed on the **t_snddis** and **t_rcvdis** routines. TCP does not allow data to be sent during reset of connection, therefore **long discon** is set to –2.
- **servtype** – Contains the T_COTS_ORD service type supported by the TCP. T_COTS_ORD is a connection-mode transport service with orderly release.

3.2.10 t_optmgmt

The **t_optmgmt** routine negotiates options with the TCP driver.

NOTE

> *This routine may only be called when the endpoint is in the TS_IDLE state. If called while in any other state, the state becomes invalid. Because of this, Sequent's implementation of TCP includes a STREAMS* **ioctl** *function called TCP_SETOPT. This* **ioctl** *function can negotiate options, but does not affect the state. TCP_SETOPT is described following the* **t_optmgmt** *description.*

The **t_optmgmt** routine has the following format:

```
int t_optmgmt(fd, req, ret)
int fd;
struct t_optmgmt *req, *ret;
```

The **t_optmgmt** structure contains the following members:

```
struct t_optmgmt {
   struct netbuf opt;  /* a struct tcp_options          */
   long flags;         /* T_CHECK, T_DEFAULT, T_NEGOTIATE */
}
```

The **opt** field is a structure of type **tcp_options**. This structure contains the following members:

```
struct tcp_options {
    ulong pr_options;   /* per protocol options           */
    int  ltime;         /* linger time (in seconds)       */
    int snd_buf;        /* send buffer space, 0= use default */
    int rcv_buf;          /* rcv buffer space, 0= use default  */
};
```

The **pr_options** field specifies the TCP options being negotiated or returned. **pr_options** can be set to one or more of the following options:

TP_DEBUG	This option turns on protocol trace debugging for this endpoint. TP_DEBUG is off by default.
TP_DONTROUTE	This option tells the TCP driver to route packets directly to the network interface, thus bypassing routing tables. Normally, this option is only used when diagnosing network problems or dealing with a poorly configured network. TP_DONTROUTE is off by default.
TP_KEEPALIVE	This option causes "keepalive" packets to be sent to the remote endpoint. This ensures that a TCP connection will remain in effect as long as the other end keeps responding. Setting this option has the additional benefit that network failures are detected faster. However, this option generates unnecessary network traffic, so use it with care. TP_KEEPALIVE is off by default.
TP_LINGER	This option allows queued data to be processed for a specified length of time after the endpoint's file descriptor is closed. The length of time this processing may continue is specified in the **ltime** field of the **tcp_options** structure. This time includes any protocol close processing. Setting this option tells TCP to use the **ltime** field. TP_LINGER is on by default.
TP_NODELAY	This option tells the TCP driver to immediately send all data, regardless of packet size, after being queued (if possible). Setting this option generates many more network packets with interactive traffic, but is useful for network window systems such as X to minimize the latency of time critical data (for example, mouse movement information). TP_NODELAY is off by default. When TP_NODELAY is off, TCP attempts to combine small packets together before sending.

TP_REUSEADDR	This option tells the TCP driver to bypass normal address matching rules when assigning an address. TP_REUSEADDR is typically used by servers that bind a wildcard address to a port number when the port may already be in use by an explicit lookup. The TP_REUSEADDR option is off by default.

The other fields in the **opt** structure are used as follows:

ltime	This field specifies the linger time in seconds. Setting **ltime** to zero tells the TCP driver not to process queued data after the endpoint is closed. Because of this, use this option with care to avoid unintentional loss of queued data. The default value is defined by TP_LINGDEF in *netinet.h*. **ltime** only has meaning when the TP_LINGER option is set. Otherwise, this field is ignored.
snd_buf	This field sets the TCP send buffer window size. The **snd_buf** and **rcv_buf** fields are typically used to increase window sizes for applications performing large data transfers. If this value is zero or invalid, the size is set to the system wide default size. You can reconfigure the system wide defaults at system build time. The maximum value is defined by TCP_MAXWIN in *netinet.h*.
rcv_buf	This field sets the TCP receive buffer window size. If this value is zero or invalid, the size is set to the system wide default size.

The action performed by **t_optmgmt** depends on the value of the **flags** field in the **req** structure. To get the current TCP options of an endpoint, set **flags** to T_DEFAULT. A T_DEFAULT call returns the current options in the **opt** field of the **ret** structure. To check whether certain options are supported by ptx/TCP/IP, specify the options in the **opt** field of **req** and set **flags** to T_CHECK. Upon return from a T_CHECK call, the **flags** field in the **ret** structure is set to either T_SUCCESS or T_FAILURE to indicate the validity of the options. Finally, to negotiate option values with the TCP driver, specify the options in the **opt** field of **req** and set **flags** to T_NEGOTIATE. The negotiated values are returned in the **opt** field of the **ret** structure.

The following three-step process should be used to change the values of any transport endpoint options:

1. Issue a T_DEFAULT call to return the current flags and parameters.
2. Modify the **ret** structure to set the values desired.
3. Issue a T_NEGOTIATE call to commit the new values.

Further, **t_optmgmt** is only allowed in the T_IDLE state. If you need to set options in other states, the TCP_SETOPT ioctl is described in the next section.

3.2.11 TCP_SETOPT

The TLI specifies that **t_optmgmt** may only be called when the endpoint is in the TS_IDLE state, which occurs after the endpoint is bound, but before a connection is established. If called while in any other state, the state becomes invalid. Some options, such as TP_REUSEADDR, are most useful before binding the endpoint and, thus, before being in the TS_IDLE state. To work around this problem, Sequent's implementation of TCP includes a STREAMS ioctl function called TCP_SETOPT.

TCP_SETOPT uses the same **tcp_options** argument as **t_optmgnt**. The function has the same effect as calling **t_optmgmt** with **flags** set to T_NEGOTIATE. However, TCP_SETOPT does not change the state or make the state invalid.

To use the function, you must include the following header files in your application:

- *<sys/types.h>*
- *<sys/stream.h>*
- *<sys/stropts.h>*
- *<netinet/netinet.h>*

The following code shows an example of using TCP_SETOPT to set the TP_KEEPALIVE and TP_LINGER options:

```
struct strioctl ioc;
struct tcp_options tcpoptbuf;
        .
        .
        .
```

```
/* set TCP options */
tcpoptbuf.pr_options = TP_KEEPALIVE | TP_LINGER;
tcpoptbuf.ltime = 60;       /* reset linger time to 1 min */
tcpoptbuf.rcv_buf = 0;      /* keep current setting */
tcpoptbuf.snd_buf = 0;      /* keep current setting */
ioc.ic_cmd = TCP_SETOPT;
ioc.ic_timout = 0;
ioc.ic_len = sizeof(struct tcp_options);
ioc.ic_dp = (char *) &tcpoptbuf;

if(ioctl(0, I_STR, &ioc) < 0)
   perror("ioctl TCP_SETOPT failed");
```

3.2.12 t_sync

The **t_sync** routine synchronizes the library data structures managed by the TCP library with information from the TCP driver and returns the state of driver. This is necessary when converting a raw file descriptor to a transport endpoint or when cooperating processes use the same endpoint. This routine has the following format:

```
int t_sync(fd)
int fd;
```

fd is the transport endpoint to be synchronized. The state returned may be one of the following:

- T_DATAXFER – The TCP driver is transferring data.
- T_IDLE – The TCP driver is idle.
- T_INCON – An incoming connection is pending.
- T_INREL – The TCP driver is waiting for an orderly release request.
- T_OUTCON – An outgoing connection is pending.
- T_OUTREL – The TCP driver is waiting for an orderly release indication.
- T_UNBND – The TCP driver is in the unbound state.

3.2.13 t_unbind

The **t_unbind** routine disables a transport endpoint placing it in the unbound state. This allows an open endpoint to be reused, rather than closing it. This routine has the following format:

```
int t_unbind(fd)
int fd;
```

The endpoint must either be a listening endpoint (server) or not actively connected.

3.3 Connection-Establishment Routines

The connection-establishment routines allow a client application to create a connection, or virtual circuit, to a server application. The connection is made through the transport endpoint and transport-provider services established during the local management phase.

The TCP connection-establishment phase consists of the following routines:

- **t_accept**
- **t_connect**
- **t_listen**
- **t_rcvconnect**

3.3.1 t_accept

The **t_accept** routine accepts a client's request for a transport connection. This routine has the following format:

```
int t_accept(fd, resfd, call)
int fd, resfd;
struct t_call *call;
```

The **t_call** structure uses the **sockaddr_in** format address and the sequence number provided by the return from **t_listen** routine.

If **fd** is not equal to **resfd**, a connection is accepted on the queue referred to by **resfd**. The queue referred to by **fd** then remains in the listen state.

If **fd** equals **resfd**, any pending temporary endpoints created by connect requests will be destroyed. The queue referred to by **fd** is then placed in a connected state.

If the value of **qlen** is greater than 0, the transport endpoint can be used to listen for connect indications. If **qlen** is greater than 1, the **t_accept** call may return with a T_LOOK event, which indicates another connect indication has been queued and is waiting to be processed. The value of **qlen** indicates the maximum number of outstanding connect indications that can be queued and pending on the endpoint.

The server must respond to each connect indication, either accepting or rejecting the request for connection. An outstanding connect indication is one to which the server has not yet responded. Often, a server will fully process a single connect indication and respond to it before receiving the next indication. In this case, a value of 1 is appropriate for **qlen**. Some servers may wish to retrieve several connect indications before responding to any of them. In such cases, a **qlen** value of 2 or more may be appropriate. However, if you set **qlen** to greater than 1, the queuing of connect indications must be handled by the server itself. TCP does not queue connect indications for you. For an example of code that queues pending connect indications, see the server example in Appendix A, "Programming Examples."

3.3.2 t_connect

The **t_connect** routine attempts to initiate a transport connection with a transport user at a specified destination. This routine has the following format:

```
int t_connect(fd, sndcall, rcvcall)
int fd;
struct t_call *sndcall, *rcvcall;
```

sndcall and **rcvcall** refer to a **t_call** structure that contains the following members:

```
struct t_call {
    struct netbuf addr;        /* a struct sockaddr_in */
    struct netbuf opt;        /* a struct tcp_options */
    struct netbuf udata;       /* unused              */
    int sequence;              /* unused              */
};
```

In nonblocking mode, a successful **t_connect** operation does not indicate that a connection has occurred, only that the attempt to perform the connection was successful. In this case, positive confirmation of the connection can only be done with the **t_rcvconnect** routine. In blocking mode, **t_connect** issues a **t_rcvconnect** call and waits for connection confirmation.

The **sockaddr_in** format address specified by the **addr** field must have a fully qualified port and IP address. If INADDR_ANY is used to specify the Internet address, it indicates "this" machine. The **opt** field refers to a **tcp_options** structure that has been modified using the values returned by a **t_optmgmt** routine.

Since TCP does not allow data to be sent as part of the connection establishment phase, the **udata** field is unused. The **sequence** field is a remnant from the **t_accept** routine's use of the **t_call** structure and is not used by **t_connect**.

3.3.3 t_listen

The **t_listen** routine allows a server to retrieve an indication of a connection request from a client. This routine has the following format:

```
int t_listen(fd, call)
int fd;
struct t_call *call;
```

The **t_call** structure contains the following members:

```
struct t_call {
  struct netbuf addr;   /* sockaddr_in address
                           of remote endpoint   */
  struct netbuf opt;    /* current options      */
  struct netbuf udata;  /* unused by TCP        */
  int sequence;         /* identifier used by
                           t_accept/t_snddis    */
 }
```

The server should read all connect requests before issuing a **t_accept** or **t_snddis** routine. The sequence number must be used during a **t_accept** or **t_snddis** routine call as they are used to match the pending kernel connection data structure.

Once a connection request has been queued in the kernel, a repeat of the request by the remote endpoint will not generate another connection indication.

t_listen is a passive routine. If no connect indications arrive, it will not return and should be considered an error. If **t_listen** is performed on an endpoint that was bound with **qlen** set to zero, it will never return unless interrupted.

3.3.4 t_rcvconnect

The **t_rcvconnect** routine is used in nonblocking mode to receive connection confirmation of a previous **t_connect** routine. This routine has the following format:

```
int t_rcvconnect(fd, call)
int fd;
struct t_call *call; /* may be NULL */
```

The **t_call** structure contains the following members:

```
struct t_call {
  struct netbuf addr;   /* sockaddr_in of remote endpoint */
  struct netbuf opt;    /* tcp_options if maxlen set      */
  struct netbuf udata;  /* not used by tcp                */
  int sequence;         /* unused                         */
}
```

t_rcvconnect should not be used on an endpoint in blocking mode because it will never return. In blocking mode, connection confirmation is handled by the **t_connect** routine.

3.4 Data Transfer Routines

The data transfer routines allow bidirectional data transfer between applications over an established connection. The TLI does not differentiate the client application from the server application during this phase. Either application can send and receive data with the TLI guaranteeing reliable, sequenced delivery over the connection created using the connection establishment routines.

The TCP data transfer phase consists of the following routines:

- **t_rcv**
- **t_snd**

3.4.1 t_rcv

The **t_rcv** routine retrieves data that has arrived over a transport connection. This routine has the following format:

```
int t_rcv(fd, buf, nbytes, flags)
int fd;
char *buf;
unsigned nbytes;
int *flags;
```

nbytes specifies the maximum number of bytes that can be gathered from a connected endpoint. The **buf** holds the received data. If the number of bytes actually received by the transport endpoint is greater than the **nbytes** argument, **flags** is set to T_MORE to indicate that more data is available to read.

TCP is a byte stream protocol and, thus, no attempt is made to preserve message boundaries. A **t_rcv** call may return less than the number of bytes requested and have additional data in transit. Applications should not depend on message size being preserved when data is sent between two endpoints. If an application requires preservation of message boundaries, this must be built into the application protocol. In general, when a **t_rcv** call returns with less than the requested number of bytes it does not indicate an end-of-file or a closed connection. The application must explicitly check for these events.

3.4.2 t_snd

The **t_snd** routine sends data over an established transport connection. This routine has the following format:

```
int t_snd(fd, buf, nbytes, flags)
int fd;
char *buf;
unsigned nbytes;
int flags;
```

The **t_snd** routine must be issued on a connected endpoint. If no space is available in the TCP transmission window, **t_snd** will block.

3.5 Connection Release Routines

The connection release routines break the transport connection created with the connection establishment routines. The routines support both an abortive and orderly release.

The TCP connection release phase consists of the following routines:

- **t_rcvdis**
- **t_rcvrel**
- **t_snddis**
- **t_sndrel**

3.5.1 t_rcvdis

The **t_rcvdis** routine returns a disconnect indication including a reason code. This routine has the following format:

```
t_rcvdis(fd, discon)
int fd;
struct t_discon *discon;
```

The **t_discon** structure contains the following members:

```
struct t_discon {
   struct netbuf udata; /* not used by TCP    */
   int reason;          /* errno.h reason     */
   int sequence;        /* if t_listen failed */
```

The cause of the disconnect may be identified by looking at the **reason** field and using *errno.h* to decode the error.

3.5.2 t_rcvrel

The **t_rcvrel** routine returns an indication that the remote user has requested an orderly release of a connection. This routine has the following format:

```
t_rcvrel(fd)
int fd;
```

Use of this routine is optional. Connection release is typically performed by issuing a **t_close** call for the local and remote endpoints.

3.5.3 t_snddis

The **t_snddis** routine aborts a connection or rejects a connect request. This routine has the following format:

```
int t_snddis(fd, call)
int fd;
struct t_call *call;
```

The **t_call** structure contains the following members:

```
struct t_call {
   struct netbuf addr;  /* unused                 */
   struct netbuf opt;   /* unused                 */
   struct netbuf udata; /* unused by TCP          */
   int sequence;        /* id of connect request */
}
```

The **t_snddis** routine will issue an immediate disconnect on the transport endpoint and reset the endpoint to an IDLE state. The **call** parameter is only required if the server rejects a connect request from a remote endpoint. The **call** parameter may be NULL if the connection is already active.

Care must be used in calling this routine as it flushes any pending data to be sent or received.

3.5.4 t_sndrel

The **t_sndrel** routine initiates an orderly release on a transmission endpoint. This routine has the following format:

```
int t_sndrel(fd)
int fd;
```

The **t_sndrel** routine tells the remote endpoint that no more data will be sent from the local endpoint. However, the local endpoint may continue to receive data from the remote endpoint until the remote endpoint issues a **t_ordrel** or a **t_close**.

4. Domain Name
Server Configuration

4. Domain Name Server Configuration

Chapter 4
User Datagram Protocol (UDP)

4.1 Introduction 4-1
4.2 Local Management Routines 4-1
t_bind 4-2
t_open 4-2
4.3 Data Transfer Routines 4-4
t_rcvudata 4-4
t_rcvuderr 4-5
t_sndudata 4-6

Chapter 4
User Datagram Protocol (UDP)

4.1 Introduction

This chapter describes the UDP routines that an application (transport user) uses to access the UDP driver (transport provider). UDP provides a connectionless-mode communication service. Two phases characterize the UDP service: local management and data transfer. Specifically, an application using UDP through the TLI will need to perform the following tasks:

- Open a transport endpoint
- Bind an endpoint to an Internet address and port
- Send and receive data
- Unbind and close an endpoint

All UDP applications must include the <*tiuser.h*> and <*tihdr.h*> header files. The *nsl* and *inet* libraries must be specified at build time. Any TLI routines using the **sockaddr_in** address format, require the <*netinet/in.h*> header file. Routines using the **tcp_options** structure require the <*netinet/netinet.h*> header.

4.2 Local Management Routines

Just as with connection-mode service, the transport users must perform appropriate local management steps before data can be transferred. A user must use the `t_open` routine to open the UDP transport endpoint and the `t_bind` routine to establish its identity.

4.2.1 t_bind

The `t_bind` routine binds an address to a given transport endpoint opened with the `t_open` routine. This routine has the following format:

```
int t_bind(fd, req, ret)
int fd;
struct t_bind *req;
struct t_bind *ret;
```

req and **ret** refer to a **t_bind** structure that contains the following members:

```
struct t_bind {
   struct netbuf addr; /* a struct sockaddr_in*/
   unsigned qlen;      /* unused in UDP */
}
```

4.2.2 t_open

The `t_open` routine opens an endpoint for a UDP connection. This routine should be called with the following arguments:

```
int t_open(path, oflag, info)
char *path;
int oflag;
struct t_info *info;
```

path should be set to TLI_UDP (defined in <*netinet/in.h*>).

oflag can be set to one or both of the following flags:

- O_RDWR – set endpoint to read and write
- O_NDELAY – set endpoint as non-blocking. All subsequent transport calls will also be non-blocking

info points to a **t_info** structure that returns UDP information. The **t_info** structure contains the following members:

```
struct t_info {
   long addr;
   long options;
   long tsdu;
   long etsdu;
   long connect;
   long discon;
   long servtype;
}
```

On a successful return, **info** contains the following fields:

- **addr** – Contains the maximum size of a transport protocol address.
- **options** – Contains the maximum number of bytes of UDP-specific options.
- **tsdu** – Contains the maximum size of a *transport service data unit.*
- **etsdu** – Set to –2 because the concept of expedited data is not supported for UDP.
- **connect** – Not relevant for UDP.
- **discon** – Not relevant for UDP.
- **servtype** – Contains the T_CLTS service type supported by UDP.

4.3 Data Transfer Routines

Once a user has bound an address to the transport endpoint, datagrams can be sent or received over that endpoint. Each outgoing message is accompanied by the address of the destination user.

Data may be transmitted over UDP by calling `t_sndudata`. Data may be received by calling `t_rcvudata`. The `t_rcvuderr` routine retrieves data transmission error packets from the network.

4.3.1 t_rcvudata

The `t_rcvudata` routine receives data from the UDP transport endpoint. This routine has the following format

```
int t_rcvudata(fd, unitdata, flags)
int fd;
struct t_unitdata *unitdata;
int *flags;
```

fd identifies the transport endpoint and **unitdata** refers to the **t_unitdata** structure that contains the following members:

```
struct t_unitdata {
   struct netbuf addr;
   struct netbuf opt;
   struct netbuf udata;
}
```

The **addr** structure specifies the sender's Internet address and port number. The **maxlen** field of **addr** must be set to at least the size of a **sockaddr_in** structure before calling the routine. The **udata** structure specifies the user buffer into which the received data is to be placed. The **maxlen** field of **udata** must be set before calling the routine to indicate the maximum amount of data that can be accepted.

If the buffer defined in the **udata** field is not large enough to hold the received data, the buffer is filled and **flags** is set to **T_MORE**. This indicates one or more additional **t_rcvudata** calls must be issued to retrieve the rest of the data. Subsequent **t_rcvudata** calls return zero in the **maxlen** field of **addr** until the full data unit has been received.

4.3.2 t_rcvuderr

The `t_rcvuderr` routine receives information concerning a possible error on a previously sent UDP datagram. It should only be issued after a T_UDERR event occurs. This routine has the following format:

```
int t_rcvuderr(fd, uderr)
int fd;
struct t_uderr *uderr;
```

fd identifies the transport endpoint and **uderr** refers to the **t_uderr** structure that contains the following members:

```
struct t_uderr {
   struct netbuf addr;
```

```
        struct netbuf opt;   /* not used   */
        long error;
    }
```

The **addr** structure contains the destination Internet address of the erroneous data unit. The **error** field specifies an error code. Possible errors are listed in **errno.h**. Other fields of the **t_uderr** structure should be ignored.

4.3.3 t_sndudata

The `t_sndudata` routine sends data from the local UDP transport endpoint. This routine has the following format:

```
    int t_sndudata(fd, unitdata)
    int fd;
    struct t_unitdata *unitdata;
```

fd identifies the transport endpoint and **unitdata** refers to a **t_unitdata** structure that contains the following members:

```
    struct t_unitdata {
        struct netbuf addr;
        struct netbuf opt;   /* not used */
        struct netbuf udata;
    }
```

The **addr** structure specifies the destination Internet address and port number. The **opt** structure is ignored as there are no UDP-specific options. The **udata** structure specifies the user data to be sent.

5. Porting Sockets to TLI

5. Porting Sockets
to TLI

Chapter 5
Porting from Sockets to the TLI

5.1 Introduction 5-1

5.2 Handling Signals 5-1

5.3 Using Buffers Allocated by t_alloc 5-2
- t_alloc Buffer Length 5-4
- t_alloc Structure Initialization 5-5
- Allocated Structures not Automatically Freed 5-5

5.4 TLI State Validation 5-5

5.5 Porting a TCP Server 5-5
- Listening for Connection Requests 5-8
- Accepting Connection Requests 5-8

5.6 Porting a TCP Client 5-9
- Client Addresses 5-11
- Connection Refusal 5-11

5.7 Using Read and Write Calls in a TLI Application 5-11

5.8 Porting a UDP Program 5-13

5.9 Porting Examples 5-14
- TCP Server Using Sockets 5-14
- TCP Server Using TLI 5-16
- TCP Client Using Sockets 5-18
- TCP Client Using TLI 5-19
- UDP Server Using Sockets 5-21
- UDP Server Using TLI 5-22
- UDP Client Using Sockets 5-24
- UDP Client Using TLI 5-25

Tables

5-1 Socket and TLI Routines for a TCP Server 5-5
5-2 Socket and TLI Routines for a TCP Client 5-10
5-3 Socket and TLI Routines for a UDP Program 5-13

Chapter 5
Porting from Sockets to the TLI

5.1 Introduction

This chapter describes what you must consider when porting a socket-based TCP application to the the Transport Layer Interface (TLI). Sockets first appeared in UNIX 4.2 BSD. Sockets provide the interface necessary to access the TCP and UDP transport services that are bundled with 4.2 BSD. The TLI was introduced in AT&T's UNIX System V Release 3 and is included in Sequent's operating system. Like sockets, the TLI provides the interface to access transport services. However, unlike sockets, the TLI was not designed for a specific transport protocol. Instead, the TLI is built upon an operating system facility called STREAMS. STREAMS provides a flexible, modular way to implement various transport protocols between the TLI and the network, while maintaining the same familiar transport interface.

Sockets and the TLI are similar in many ways. Both require the same general steps for interprocess communication to occur. Many socket and TLI routines have similar names and functions. There are also a number of differences. For example, TLI requires you to allocate certain data structures for use with TLI routines, while sockets typically do not. TLI also enables servers to turn down a connection request before establishing the connection, while sockets require all connection requests to be honored.

The remaining sections discuss the similarities and differences between socket and TLI-based TCP and UDP programs.

5.2 Handling Signals

In socket-based applications, **ioctl**, **read**, and **open** calls are restarted automatically after a signal is processed. However, TLI routines are not restarted after signal processing. Thus, a TLI routine may fail and return a value of EINTR if a signal occurs after the routine is invoked. TLI applications should block signals (using **sighold**) when issuing a **t_accept** call. When issuing other TLI calls, applications should either block signals or check for a return value of EINTR and restart the call if the error occurs.

NOTE

For the sake of simplicity, the examples in this chapter do not show signal blocking. However, signal blocking is shown in the examples in Appendix A. Once you have studied and understood the porting examples, you may want to study the more complex examples in Appendix A.

5.3 Using Buffers Allocated by t_alloc

The netbuf structure allocated by **t_alloc** is defined in `/usr/include/sys/tiuser.h`:

```
struct netbuf {
        unsigned int maxlen;
        unsigned int len;
        char *buf;
};
```

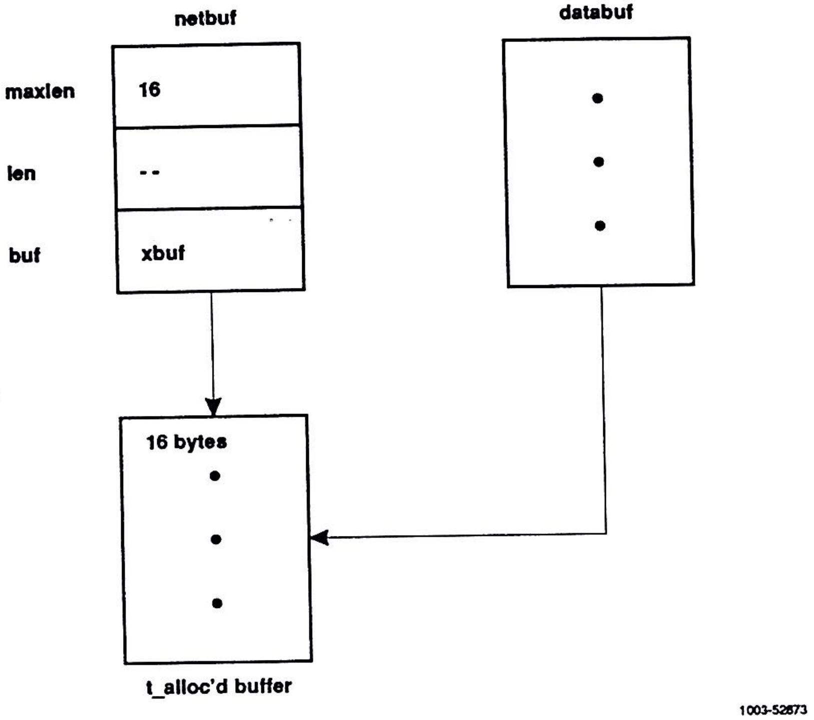
If you're writing an application from scratch, using TLI, and you use `t_alloc`, you do not need to use additional buffers for data transfer, addresses, and so forth. The buf pointer from the netbuf structure points to the buffer you should use. For example, if the application reads directly from user input, your program could read directly into the netbuf buffer.

When porting socket code, it is common to have additional static buffers allocated for storing data, addresses, and so forth. If you're using static buffers, you must use **memcopy** to move the data from the static buffer to the netbuf buffer.

The following is an example of using **memcopy**.

```
memcopy(data->udata.buf, (char *)&databuf,
         sizeof(databuf))
```

Using memcopy results in the following actions:

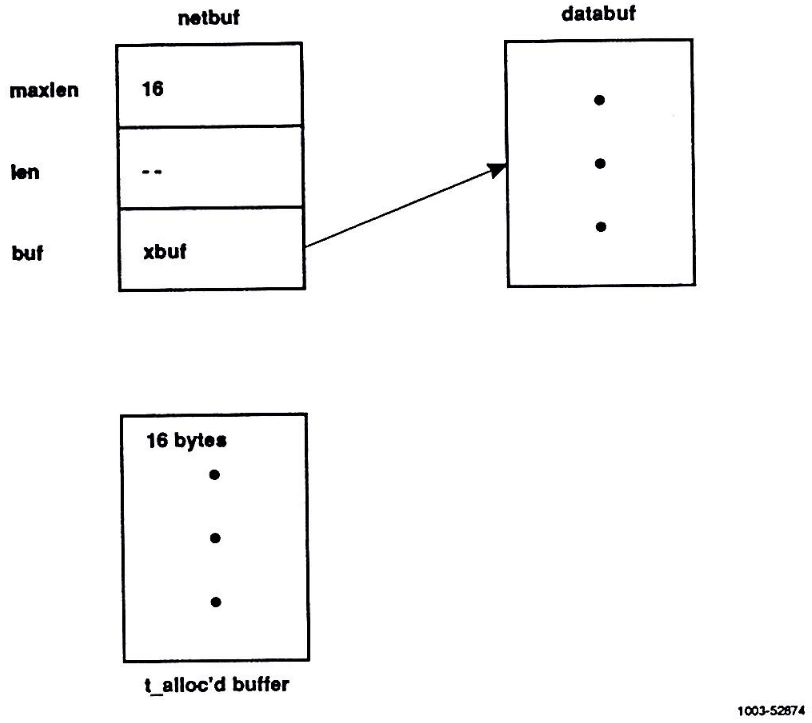

netbuf
databuf
maxlen
16
len
--
buf
xbuf
16 bytes
t_alloc'd buffer
1003-52673

A common mistake when using netbuf and static buffers is to assign the netbuf pointer to point to the static buffer. An assignment is correct in the C language, but when a program later tries to **t_free** the buffer, the pointer no longer points to the buffer that was allocated by **t_alloc**.

In the following example, assigning pointers results in an unfreeable buffer.

```
data->udata.buf = (char *)&databuf;
```

This results in the following:


netbuf
databuf
maxlen
16
len
--
buf
xbuf
16 bytes
t_alloc'd buffer
1003-52874

t_free will not be able to free the allocated buffer. In a short-lived client application, this may not result in problems. In servers, this type of error could result in the process growing unbounded and consuming system memory.

5.3.1 t_alloc Buffer Length

t_alloc always allocates a max length buffer. If the underlying transport provider supports large packets (16K, for example), and the application wants to send a one byte packet, using an allocated buffer could be inefficient.

5.3.2 t_alloc Structure Initialization

Your program must initialize the len element of the netbuf structure. TLI calls do not validate the maxlen value until after len is used. For some structures allocated with **t_alloc** maxlen will be set to 0 by the provider, however, len will be uninitialized. This could result in the user process dumping core if the user process does not also initialize len to 0.

5.3.3 Allocated Structures not Automatically Freed

The **t_close** does not free buffers that were associated with the closing endpoint. The user application must **t_free** them.

5.4 TLI State Validation

TLI state vaildation does not work properly. For example, a process can call **t_send** in the T_IDLE state and no error will result. The process state should actually look for T_DATAXFER. Do not rely on TLI to do state validation.

5.5 Porting a TCP Server

Socket and TLI-based servers perform the same major actions to establish a connection between the server and client, and then provide the requested service. Table 5-1 lists the major actions performed by a TCP server and the socket and TLI routines used to perform those actions. In general, the actions are shown in the order in which they should occur, although some actions may occur more than once within a program. The sections following the table describe situations where socket and TLI-based servers differ.

Table 5-1
Socket and TLI Routines for a TCP Server

Action	Socket Routines	TLI Routines
Create an endpoint	**socket**	**t_open**

Allocate memory for **t_bind** data structures (Used by **t_bind** call)		**t_alloc**
Bind an Internet address to the endpoint	**bind**	**t_bind** (qlen > 0)
Allocate memory for **t_call** data structure (Used by **t_listen** and **t_accept** calls)		**t_alloc**
Listen for an incoming connection request	**listen**	**t_listen**
Reject a connection request		**t_snddis**

Table 5-1
Socket and TLI Routines for a TCP Server (cont.)

Action	Socket Routines	TLI Routines
Accept a connection request	**accept**	Issue **t_open** and **t_bind** for an alternate endpoint, then issue **t_accept**[1]
Wait for an event	**select**	**poll**
Receive data from a client over a connection	**read** **recv** **recvmsg**	**read**[2] **t_rcv**
Send data to a client over a connection	**write** **send** **sendmsg**	**write**[2] **t_snd**
Free memory used by TLI data structures (For structures created with **t_alloc**)		**t_free**
Check for orderly closure request		**t_rcvrel**
Release a connection (orderly closure)	**close** **shutdown**	**t_sndrel** **t_close**
Release a connection (abrupt closure)	**shutdown**	**t_snddis,** **t_close** (linger=0)
Unbind an endpoint from an address	**unlink**	**t_unbind.**

1. An application can accept on the same endpoint used to listen for the connection request, although this is not recommended.
2. Requires that the special **tirdwr** module be pushed onto the TCP stream. For details on using **tirdwr** and the read/write interface, see the *DYNIX/ptx Network Programmer's Guide.*

5.5.1 Listening for Connection Requests

A TLI server can choose whether it wants to honor a connection request from a client before it actually accepts the request. With sockets, the server must accept a connection request before it can reject it.

In sockets, the **listen** routine marks the socket as listening and returns; the server must then call **accept** to be notified of an incoming connection request. The **accept** routine blocks until a connection request arrives. When **accept** returns, a connection has been established. The only way to refuse a connection is to accept it and then immediately close the connection.

On the other hand, TLI's **t_listen** routine normally blocks and only returns when a connection request arrives. The server can then decide whether it wants to accept the request. The server calls **t_accept** to accept the connection or **t_snddis** to reject it. Whether an endpoint is a server is based on the value of the **qlen** field in the **t_bind** routine.

5.5.2 Accepting Connection Requests

When a server chooses to accept a connection request, it usually creates a new transport endpoint to handle further communication between the client and the server. This is true in both sockets and the TLI. However, the two interfaces differ in the way they create the new endpoint.

In sockets, the **accept** routine returns a new endpoint (socket) each time it is called. The server passes the new endpoint to a new process that is spawned specifically to handle the connection. The server continues to listen on the original endpoint for more connection requests.

In the TLI, the server uses the **t_open** and **t_bind** routines to create a new endpoint. The original endpoint and the new endpoint are both used as parameters to the **t_accept** routine. Then, as with sockets, the new endpoint is passed to a new process that is spawned to handle the new connection.

The **t_accept** call may also return T_LOOK, which indicates a new connect indication has arrived since the last **t_listen** call. In this case, **t_listen** must be used to retrieve the indication before any **t_accept** call will succeed. Connect indications that arrive before they can be processed should be queued for later processing. The maximum number of connect indications that can be pending is specified by the **qlen** field of the **t_bind** routine. For an example of queuing connect indications, see the server example in Appendix A, "Programming Examples."

5.6 Porting a TCP Client

Socket and TLI-based clients also perform the same major actions. They differ only in the way their routines perform those actions. Table 5-2 lists the major actions performed by a TCP client and the socket and TLI routines used to perform those actions. In general, the actions are shown in the order in which they should occur, although some actions may occur more than once within a program. The sections following the table describe situations where socket and TLI-based clients differ.

Table 5-2
Socket and TLI Routines for a TCP Client

Action	Socket Routines	TLI Routines
Create an endpoint	**socket**	**t_open** (path= TLI_TCP)
Allocate memory for **t_bind** data structures (Used by **t_bind** call)		**t_alloc**
Bind an Internet address to the endpoint	Done as part of **connect**	**t_bind** (qlen=0)
Allocate memory for **t_call** data structure (Used by **t_connect** and **t_rcvconnect** calls)		**t_alloc**
Place a connection request	**connect**	**t_connect**
Confirm connection was made (in nonblocking mode)		**t_rcvconnect**
Send data to a server over a connection	**write** **send** **sendmsg**	**write**[1] **t_snd**
Receive data from a server over a connection	**read** **recv** **recvmsg**	**read**[1] **t_rcv**
Free memory used by TLI data structures (For structures created with **t_alloc**)		**t_free**
Release a connection (orderly closure)	**close** **shutdown**	**t_sndrel** **t_close**
Release a connection (abrupt closure)	**shutdown**	**t_snddis** **t_close**
Unbind an endpoint from an address	**unlink**	**t_unbind**

1. Requires that the special **tirdwr** module be pushed onto the TCP stream. For details on using **tirdwr** and the read/write interface, see the *DYNIX/ptx Network Programmer's Guide.*

5.6.1 Client Addresses

Clients typically don't care about their own addresses, since no other client will try to connect with them. Thus, with sockets, a client often doesn't call **bind**. However, TLI clients still must call **t_bind** and explicitly specify a NULL address. In both cases, TCP chooses an address and assigns the endpoint to it.

5.6.2 Connection Refusal

With sockets, a call to **connect** cannot indicate a connection refusal. For a socket-based server to refuse a connection request and a client to detect that refusal, the connection must first be established and then released with the **close** or **shutdown** routines.

On the other hand, a TLI client's **t_connect** (in blocking mode) or **t_rcvconnect** (in nonblocking mode) routines can return with an indication that the server rejected the connection request.

5.7 Using Read and Write Calls in a TLI Application

Sockets provide several routines that can transfer data between two endpoints. Two of those routines are **read** and **write**—the same calls used to read and write ordinary disk files. TLI applications can also use **read** and **write**, but special preparation is required before using them.

To use the **read** and **write** routines, a TLI application must push the **tirdwr** module onto the TCP stream. The module is pushed with the STREAMS' **ioctl** I_PUSH call. The **tirdwr** module should only be pushed when the transport endpoint is in the data transfer phase. This is because **tirdwr** effectively replaces the standard TLI interface with a pure read/write interface. Once **tirdwr** is pushed, you cannot make any **t_*** calls on the steream. No TLI routines can be called while **tirdwr** is on the TCP stream. Furthermore, popping **tirdwr** off the TCP stream will close the transport connection. Thus, once **tirdwr** is pushed on the stream, you must use **close** when you want to release the connection.

For example:

```
if(ioctl(fd, I_PUSH, "tirdwr") < 0) {
    perror("I_PUSH of tirdwr on fd failed");
    exit(1);
}
```

To use this ioctl, your program needs to include `/usr/include/sys/ioctl.h`

For more details on using the **tirdwr** module and the read/write interface, see the *DYNIX/ptx Network Programmer's Guide.*

5.8 Porting a UDP Program

Socket and TLI-based UDP applications perform the same major actions during connection-less mode communication. Table 5-3 lists the major actions performed by a UDP application and the socket and TLI routines used to perform those actions. In general, the actions are shown in the order in which they should occur, although some actions may occur more than once within a program.

Table 5-3
Socket and TLI Routines for a UDP Program

Action	Socket Routines	TLI Routines
Create an endpoint	**socket**	**t_open** (path= TLI_UDP)
Allocate memory for **t_bind** data structures (Used by **t_bind** call)		**t_alloc**
Bind an Internet address to the endpoint	**bind**	**t_bind**
Allocate memory for **t_unitdata** data structure		**t_alloc**
Send data to another UDP process	**sendto**	**t_sndudata**
Receive data from another UDP process	**recvfrom**	**t_rcvudata**
Wait for an event	**select**	**poll**
Free memory used by TLI data structures (For structures created with **t_alloc**)		**t_free**
Unbind an endpoint from an address	**unlink**	**t_unbind**

5.9 Porting Examples

This section contains simple examples that show socket-based TCP and UDP programs and equivalent programs ported to the TLI. Each example shows the major routines used for a particular type of TCP or UDP program. The TCP client examples also illustrate use of the **gethostbyname** routine to determine the IP address of a host providing a service. By comparing the original socket-based code and the resulting TLI-based code, you can see what changes must be made to port TCP and UDP applications.

All the examples allow TCP to assign port numbers, rather than specifying predefined ports. For examples of using predefined port numbers for a TCP or UDP service, see the examples in Appendix A.

All examples are supplied on the ptx/TCP/IP tape. After the tape is installed, the examples can be found in the */usr/sequent/tcp_examples* directory. This directory includes a makefile you can use to build all the TLI examples in this manual.

5.9.1 TCP Server Using Sockets

```
/* tcp_sock_server.c */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

/*
 * This program creates a socket and begins an infinite loop.
 * Each time through the loop it accepts a connection and prints
 * out messages from it.  When the connection breaks, or a termination
 * message comes through, the program accepts a new connection.
 */

main()
{
        int sock, length;
        struct sockaddr_in server;
        int msgsock;
        char buf[1024];
        int rval;
        int i;

        /*
         * Create socket
         */
```

```
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock < 0) {
    perror("opening stream socket");
    exit(1);
}
/*
 * Name socket using wildcards
 */
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = 0;
if (bind(sock, &server, sizeof(server))) {
    perror("binding stream socket");
    exit(1);
}
/*
 * Find out assigned port number and print it out
 */
length = sizeof(server);
if (getsockname(sock, &server, &length)) {
    perror("getting socket name");
    exit(1);
}
printf("Socket has port #%d\n", ntohs(server.sin_port));
fflush(stdout);
/*
 * Start accepting connections
 */
listen(sock, 1);
do {
    msgsock = accept(sock, 0, 0);
    if (msgsock == -1)
        perror("accept");
    else do {
        bzero(buf, sizeof(buf));
        if ((rval = read(msgsock, buf, 1024)) < 0)
            perror("reading stream message");
        i = 0;
        if (rval == 0)
            printf("Ending connection\n");
        else {
            printf("-->%s\n", buf);
            fflush(stdout);
        }
    } while (rval != 0);
    close(msgsock);
} while (1);
/*
 * Since this program has an infinite loop, the socket "sock" is
 * never explicitly closed.  However, all sockets will be closed
```

```
	 * automatically when a process is killed or terminates normally.
	 */
}
```

5.9.2 TCP Server Using TLI

```
/* tcp_tli_server.c */

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/tiuser.h>
#include <sys/tihdr.h>
#include <fcntl.h>
#include <stdio.h>
#include <netinet/in.h>
#include <netinet/netinet.h>
#include <netdb.h>

/*
 * This program creates an endpoint and begins an infinite loop.
 * Each time through the loop it accepts a connection and prints out
 * messages from it.  When the connection breaks, or a termination
 * message comes through, the program accepts a new connection.
 */

main()
{
	int fd, length, resfd;
	struct sockaddr_in *server, raddr;
	struct t_call *call;
	struct t_bind *bindreq, *bindret;
	int msgsock;
	char buf[1024];
	int maxframe = 1024;
	int flags;
	int rval;
	int ret;
	int i;

	/*
	 * Create endpoint
	 */
	if ((fd = t_open(TLI_TCP, O_RDWR, NULL)) < 0)  {
		t_error("ERROR: t_open failed for listen_fd");
		exit(1);
	}
	/*
	 *  Allocate memory for two t_bind structures.
	 */
```

```
if ((bindreq = (struct t_bind *)t_alloc(fd, T_BIND, T_ALL)) == NULL)  {
    t_error("ERROR: t_alloc of t_bind req structure failed");
    exit(1);
}
if ((bindret = (struct t_bind *)t_alloc(fd, T_BIND, T_ALL)) == NULL)  {
    t_error("ERROR: t_alloc of t_bind ret structure failed");
    exit(1);
}
/*
 * Let TCP pick an address and port
 */
server = (struct sockaddr_in *) bindreq->addr.buf;
server->sin_family = AF_INET;
server->sin_addr.s_addr = INADDR_ANY;
server->sin_port = 0;
bindreq->addr.len = sizeof(struct sockaddr_in);
bindret->addr.maxlen = sizeof(struct sockaddr_in);
bindreq->qlen = 1;
if (t_bind(fd, bindreq, bindret) < 0)  {
    t_error("ERROR: binding endpoint");
    exit(1);
}
/*
 * Find out assigned port number and print it out
 */
memcpy(&raddr, bindret->addr.buf, sizeof(struct sockaddr_in));
printf("Endpoint has port #%d\n", ntohs(raddr.sin_port));
/*
 * Start accepting connections
 */
if ((call = (struct t_call *)t_alloc(fd, T_CALL, T_ALL)) == NULL)  {
    t_error("ERROR: t_alloc of t_call structure failed");
    exit(1);
}
do {
while(((ret = t_listen(fd, call)) < 0)) {}
if ((resfd = t_open(TLI_TCP, O_RDWR, (struct t_info *)NULL)) < 0)  {
    t_error("ERROR: t_open for responding fd failed");
    exit(1);
}
if (t_bind(resfd, NULL, NULL) < 0)  {
    t_error("ERROR: t_bind for responding fd failed");
    exit(1);
}
if (t_accept(fd, resfd, call) < 0)  {
    t_error("ERROR: t_accept for responding fd failed");
    exit(1);
}
else do {
    if ((rval = t_rcv(resfd, buf, maxframe, &flags)) < 0)  {
      t_error("ERROR: reading stream message");
      exit(1);
       }
       else  {
      if (rval == 0)
          printf("Ending connection\n");
      else
          printf("-->%s\n", buf);
       }
   }  while (flags & T_MORE);
   if (t_close(resfd) < 0)  {
        t_error("ERROR: t_close failed for resfd");
        exit(1);
   }
    } while (1);
/*
 * Since this program has an infinite loop, the endpoint "fd" is
 * never explicitly closed.  However, all endpoints will be closed
 * automatically when a process is killed or terminates normally.
 */
 }
```

5.9.3 TCP Client Using Sockets

```
/* tcp_sock_client.c */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

#define DATA "Half a league, half a league . . ."

/*
 * This program creates a socket and initiates a connection with the socket
 * given in the command line.  One message is sent over the connection and
 * then the socket is closed, ending the connection.  The form of the command
 * line is:
 *              sock_client <hostname> <portnumber>
 */

main(argc, argv)
     int argc;
     char *argv[];
{
     int fd;
     struct sockaddr_in server;
     struct hostent *hp, *gethostbyname();
```

```
    char buf[1024];

    /*
     * Create socket
     */
    fd = socket(AF_INET, SOCK_STREAM, 0);
    if (fd < 0) {
        perror("opening stream socket");
        exit(1);
    }
    /*
     * Connect socket using name specified by command line
     */
    server.sin_family = AF_INET;
    hp = gethostbyname(argv[1]);
    if (hp == 0) {
        fprintf(stderr, "%s: unknown host\n", argv[1]);
        exit(1);
    }
    bcopy(hp->h_addr, &server.sin_addr, hp->h_length);
    server.sin_port = htons(atoi(argv[2]));
    if (connect(fd, &server, sizeof(server)) < 0) {
        perror("connecting stream socket");
        exit(1);
    }
    if (write(fd, DATA, sizeof(DATA)) < 0)
        perror("writing on stream socket");
    close(fd);
}
```

5.9.4 TCP Client Using TLI

```
/* tcp_tli_client.c */

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/tiuser.h>
#include <sys/tihdr.h>
#include <fcntl.h>
#include <stdio.h>
#include <netinet/in.h>
#include <netinet/netinet.h>
#include <netdb.h>

#define DATA "Half a league, half a league . . ."

/*
 * This program creates an endpoint and initiates a connection with the host
 * port given in the command line.  One message is sent over the connection,
 * then the endpoint is closed, ending the connection.  The form of the comman
 * line is:
 *              tli_client <hostname> <portnumber>
 */

main(argc, argv)
    int argc;
    char *argv[];
{
    int fd;
    struct sockaddr_in *server;
    struct hostent *hp, *gethostbyname();
    struct t_call *sndcall;
    char buf[1024];

    /*
     * Create endpoint
     */
    if ((fd = t_open(TLI_TCP, O_RDWR, NULL)) < 0) {
        t_error("ERROR: opening endpoint");
        exit(1);
    }
   /*
    *  Bind the endpoint to a local IP address.
    */
    if (t_bind(fd, NULL, NULL) < 0) {
        t_error("ERROR: t_bind failed");
        exit(1);
    }
    /*
     *  Allocate memory for a t_call structure.  The structure is
     *  used in the following t_connect call.
     */
    if ((sndcall = (struct t_call *)t_alloc(fd, T_CALL, T_ALL)) == NULL) {
        t_error("ERROR: t_alloc failed");
        exit(1);
    }
    hp = gethostbyname(argv[1]);
    if (hp == 0) {
        fprintf(stderr, "%s: unknown host", argv[1]);
        exit(1);
    }
    server = (struct sockaddr_in *) sndcall->addr.buf;
    server->sin_family = AF_INET;
    server->sin_addr.s_addr = *(unsigned long *)hp->h_addr;
    server->sin_port = htons(atoi(argv[2]));
    sndcall->addr.len = sizeof(struct sockaddr_in);
    /*
     * Connect endpoint using name specified by command line.
     */
    if (t_connect(fd, sndcall, NULL) < 0) {
        t_error("ERROR: t_connect failed");
        exit(1);
    }
    if (t_snd(fd, DATA, sizeof(DATA), 0) < 0)
        t_error("ERROR: writing on connection");
    if (t_close(fd) < 0) {
        t_error("ERROR: t_close failed");
        exit(1);
    }
    exit(0);
}
```

5.9.5 UDP Server Using Sockets

```
/* udp_sock_client.c */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>

/*
 * This program creates a datagram socket, binds a name to it, then reads
 * from the socket.
 */

main()
{
    int sock, length;
    struct sockaddr_in name;
    char buf[1024];

    /*
     * Create socket from which to read
     */
    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock < 0) {
        perror("opening datagram socket");
        exit(1);
    }
    /*
     * Create name with wildcards
     */
    name.sin_family = AF_INET;
    name.sin_addr.s_addr = INADDR_ANY;
    name.sin_port = 0;
    if (bind(sock, &name, sizeof(name))) {
        perror("binding datagram socket");
        exit(1);
    }
    /*
     * Find assigned port value and print it out
     */
    length = sizeof(name);
    if (getsockname(sock, &name, &length)) {
        perror("getting socket name");
        exit(1);
    }
    printf("Socket has port #%d\n", ntohs(name.sin_port));
    fflush(stdout);
    /*
     * Read from the socket
     */
    if (read(sock, buf, 1024) < 0)
        perror("receiving datagram packet");
    printf("-->%s\n", buf);
    close(sock);
}
```

5.9.6 UDP Server Using TLI

```
/* udp_tli_server.c */

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/tiuser.h>
#include <sys/tihdr.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <netinet/netinet.h>
#include <netdb.h>
#include <stdio.h>

/*
 * This program creates a datagram endpoint, binds a name to it, then reads
 * from the endpoint.
 */

main()

{
    int fd, length, *flags;
    struct sockaddr_in name, raddr;
    struct t_bind *bindreq, *bindret;
    struct t_unitdata *req;
    char buf[1024];
```

```
/*
 * Create an endpoint from which to read
 */
if ((fd = t_open(TLI_UDP, O_RDWR, NULL)) < 0) {
    t_error("ERROR: opening datagram endpoint");
    exit(1);
}
/*
 * Create name with wildcards.
 */
name.sin_family = AF_INET;
name.sin_addr.s_addr = INADDR_ANY;
name.sin_port = 0;
/*
 *  Allocate memory for two t_bind structures.
 */
if ((bindreq = (struct t_bind *)t_alloc(fd, T_BIND, T_ALL)) == NULL) {
    t_error("ERROR: t_alloc of t_bind bindreq structure failed");
    exit(1);
}
if ((bindret = (struct t_bind *)t_alloc(fd, T_BIND, T_ALL)) == NULL) {
    t_error("ERROR: t_alloc of t_bind bindret structure failed");
    exit(1);
}
memcpy(bindreq->addr.buf, &name, sizeof(struct sockaddr_in));
bindreq->addr.len = sizeof(struct sockaddr_in);
bindret->addr.maxlen = sizeof(struct sockaddr_in);
bindreq->qlen = 0;
if (t_bind(fd, bindreq, bindret) < 0) {
    t_error("ERROR: binding datagram fd");
    exit(1);
}
/*
 * Find assigned port value and print it out
 */
memcpy(&raddr, bindret->addr.buf, sizeof(struct sockaddr_in));
printf("Endpoint has port #%d\n", ntohs(raddr.sin_port));
/*
 * Allocate memory for struct t_unitdata structure
 */
if ((req = (struct t_unitdata *) t_alloc(fd, T_UNITDATA, T_ALL)) == NULL) {
    t_error("ERROR: t_alloc of t_unitdata structure failed");
    exit(1);
}
/*
 * Read from the endpoint
 */
if (t_rcvudata(fd, req, &flags) < 0)
    t_error("ERROR: receiving datagram packet");
printf("-->%s\n", req->udata.buf);
```

```
    if (t_close(fd) < 0)
        t_error("ERROR: t_close failed");
    exit(0);
}
```

5.9.7 UDP Client Using Sockets

```
/* udp_sock_client.c */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

#define DATA "The sea is calm tonight, the tide is full ..."

/*
 * A datagram is sent to a receiver whose name comes from the command
 * line arguments.  The form of the command line is:
 *
 *       sock_write <hostname> <portnumber>
 *
 */

main(argc, argv)

int argc;
char *argv[];

{
    int sock;
    struct sockaddr_in name;
    struct hostent *hp, *gethostbyname();
    /*
     * Create socket on which to send
     */
    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock < 0) {
        perror("opening datagram socket");
        exit(1);
    }
    /*
     * Construct name, with no wildcards, of the socket to send to.
     * Gethostbyname() returns a structure including the network address
     * of the specified host.  The port number is taken from the command
     * line.
     */
    if ((hp = gethostbyname(argv[1])) == 0) {
            fprintf(stderr, "%s: unknown host", argv[1]);
            exit(1);
        }
        bcopy(hp->h_addr, &name.sin_addr, hp->h_length);
        name.sin_family = AF_INET;
        name.sin_port = htons(atoi(argv[2]));
        /*
         * Send message
         */
        if (sendto(sock, DATA, sizeof(DATA), 0, &name, sizeof(name)) < 0)
            perror("sending datagram message");
        close(sock);
}
```

5.9.8 UDP Client Using TLI

```
/* udp_tli_client.c */

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/tiuser.h>
#include <sys/tihdr.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <netinet/netinet.h>
#include <netdb.h>
#include <stdio.h>

#define DATA "The sea is calm tonight, the tide is full ..."

/*
 * A datagram is sent to a receiver whose name comes from the command
 * line arguments.  The form of the command line is:
 *
 *       tli_write <hostname> <portnumber>
 */

main(argc, argv)

int argc;
char *argv[];

{
    int fd;
    struct sockaddr_in name;
    struct hostent *hp, *gethostbyname();
    struct t_unitdata *req;

    /*
```

```
         * Create an endpoint on which to send.
         */
        if ((fd = t_open(TLI_UDP, O_RDWR, NULL)) < 0) {
            t_error("ERROR: opening datagram endpoint");
            exit(1);
        }
        if (t_bind(fd, NULL, NULL) < 0) {
            t_error("ERROR: t_bind failed");
            exit(1);
        }
        /*
         * Construct name, with no wildcards, of the endpoint to send to.
         * Gethostbyname() returns a structure including the network address
         * of the specified host.  The port number is taken from the command
         * line.
         */
        if ((hp = gethostbyname(argv[1])) == 0) {
            fprintf(stderr, "%s: unknown host\n", argv[1]);
            exit(1);
        }
        name.sin_addr.s_addr = *(unsigned long *)hp->h_addr;
        name.sin_family = AF_INET;
        name.sin_port = htons(atoi(argv[2]));
        /*
         * Allocate memory for struct t_unitdata structure
         */
        if ((req = (struct t_unitdata *) t_alloc(fd, T_UNITDATA, T_ALL)) == NULL) {
            t_error("ERROR: t_alloc of t_unitdata structure failed");
            exit(1);
        }
        memcpy(req->addr.buf, &name, sizeof(struct sockaddr_in));
        req->addr.len = sizeof(struct sockaddr_in);
        /*
         * Set up data for transmission
         */
        strcpy(req->udata.buf, DATA);
        req->udata.len = strlen(DATA);
        /*
         * Send message
         */
        if (t_sndudata(fd, req) < 0) {
            t_error("ERROR: sending datagram message");
        }
        if (t_close(fd) < 0)
            t_error("ERROR: t_close failed");
}
```

A. Programming Examples

A. Programming Examples

Appendix A
Programming Examples

A.1 Introduction A-1

A.2 TCP Programming Examples A-1
  TCP Client Program A-1
  TCP Server Program A-4

A.3 UDP Programming Examples A-14
  UDP Client Example A-14
  UDP Server Example A-17

Appendix A
Programming Examples

A.1 Introduction

This appendix shows TCP and UDP programming examples. The examples illustrate the sequence of actions that must occur in particular types of TCP and UDP applications and the routines used to accomplish those actions.

All examples are supplied on the ptx/TCP/IP tape. After the tape is installed, the examples can be found in the */usr/sequent/tcp_examples* directory. This directory includes a makefile you can use to build all the TLI examples in this manual.

A.2 TCP Programming Examples

Two programming examples are shown here to illustrate the addressing method that must be used to drive TCP across the TLI. They show a client and a server program.

A.2.1 TCP Client Program

The client program opens an endpoint, binds to a null address (supplied by TCP), and attempts to connect to the example TCP server. When a connection is established, the client sends data to the server. The server receives the data and sends it back to the client. After sending 20 messages to the server, the client initiates an orderly release of the connection and exits.

```
/* tcp_client.c */

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/tiuser.h>
#include <sys/tihdr.h>
#include <stropts.h>
#include <fcntl.h>
#include <stdio.h>
#include <signal.h>
#include <netinet/in.h>
#include <netinet/netinet.h>
#include <netdb.h>
```

```
#define EXAMPLE_SRV  "example"
#define SERVER_HOST "exm_host"

main()
{
    int fd;
    int nbytes;
    int flags = 0;
    int len = 0;
    int exit_flag;
    int message_rcvd;
    int i;
    unsigned char buf[1024];
    struct t_call *sndcall;
    struct sockaddr_in saddr;
    struct hostent *hostptr;
    struct servent *servptr;
    extern int t_errno;

    /*
     *  Open a read/write TCP endpoint.
     */
    if ((fd = t_open(TLI_TCP, O_RDWR, NULL)) < 0) {
        t_error("ERROR: t_open failed");
        exit(1);
    }
    /*
     *  Bind the endpoint to a local IP address.
     */
    if (t_bind(fd, NULL, NULL) < 0) {
        t_error("ERROR: t_bind failed");
        exit(1);
    }
    /*
     *  Allocate memory for a t_call structure.  The structure is
     *  used in the following t_connect call.
     */
    if ((sndcall = (struct t_call *)t_alloc(fd, T_CALL, T_ALL)) == NULL) {
        t_error("ERROR: t_alloc failed");
        exit(1);
    }
    /*
     *  Get the Internet address of the example host.
     */
    if ((hostptr = gethostbyname((char *)SERVER_HOST)) == NULL) {
        t_error("ERROR: gethostbyname failed");
        exit(1);
    }
    /*
     *  Get the port number of the example server.
```

```
 */
if ((servptr = getservbyname((char *)EXAMPLE_SRV, NULL)) == NULL) {
    t_error("ERROR: getservbyname failed");
    exit(1);
}
/*
 *  Set up the t_call structure information.
 */
saddr.sin_addr.s_addr = *(unsigned long *)hostptr->h_addr;
saddr.sin_port = servptr->s_port;
saddr.sin_family = AF_INET;
memcpy(sndcall->addr.buf, (char *)&saddr, sizeof(struct sockaddr_in));
sndcall->addr.len = sizeof(struct sockaddr_in);
/*
 *  Attempt to connect to the example server.
 */
if (t_connect(fd, sndcall, NULL) < 0) {
    t_error("ERROR: t_connect failed");
    exit(1);
}
exit_flag = i = 0;
while (!exit_flag) {
    if (i < 20) {
        /*
         *  Prepare data to be sent to the server.
         */
        strcpy(buf, "Hello, world");
        i++;
    }
    else {
        /*
         *  After 20 messages have been sent,
         *  exit the loop.
         */
        exit_flag++;
    }
    len = strlen(buf);
    printf("Client: Sending %d bytes\n", len);
    /*
     *  Send data to the server.
     */
    if (t_snd(fd, buf, len, 0) < 0) {
        t_error("ERROR: t_snd failed");
        exit(1);
    }
    else
        printf("Client: Send successful. Waiting for rcv.\n");
    if (!exit_flag) {
        /*
         * Receive data from the server.
```

```
             */
            if ((nbytes = t_rcv(fd, buf, 1024, &flags)) != -1) {
                printf("Client: Rcv'd %d bytes.\n", len);
                printf("Client: Rcv string: %s\n", buf);
            }
            else {
                t_error("ERROR: t_rcv failed");
                exit(1);
            }
        }
    }
    /*
     *  Close the connection.
     */
    if (t_close(fd) < 0) {
   t_error("ERROR: t_close failed");
   exit(1);
    }
    exit(0);
}
```

A.2.2 TCP Server Program

The server program opens an endpoint, binds to a specified address, and listens for remote connection requests. When the server receives a connection request, it creates another endpoint, spawns a new process to handle the new endpoint, and listens for other connection requests. The spawned process receives messages through the new endpoint and sends copies of the messages back to the client. When the client requests an orderly release, the spawned process releases the connection and exits. The original server process continues in an infinite loop, waiting to respond to other connection requests.

```
/* tcp_server.c */

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/tiuser.h>
#include <sys/tihdr.h>
#include <stropts.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#include <netinet/in.h>
#include <netinet/netinet.h>

#define  DISCONNECT            -1
#define  LISTEN_STATE           0
```

```
#define  OPEN_RES_STATE         1
#define  BIND_RES_STATE         2
#define  ACCEPT_CALL_STATE      3
#define  ERROR_STATE            4

struct call_list {
    struct call_list *next;
    struct t_call *call;
    int resfd;
};
struct call_list *first_call;
struct call_list *end_call;

char *malloc();
pid_t fork();
pid_t wait();

extern int errno;
extern int t_errno;
extern char *optarg;
int reapchild();
char str[132];
int srv_port = 1090;

main(argc, argv)
    int argc;
    char **argv;
{
    int listen_fd;
    int conn_fd;
    struct t_bind *bindreq, *bindret;
    struct sockaddr_in saddr, raddr;
    struct sockaddr_in calladdr;
    int maxframe = 1024;
    unsigned int qlen = 2;
    int ret;
    int f;
    struct call_list *call_list_entry;
    struct call_list *tmp_call_ptr;
    int call_state;
    /************************************************************/

    sigset(SIGCLD, reapchild);

    if ((listen_fd = t_open(TLI_TCP, O_RDWR, NULL)) < 0) {
        t_error("ERROR: t_open failed for listen_fd");
        exit(1);
    }

    /*
```

```
     * Allocate memory for two t_bind structures
     */
    if ((bindreq = (struct t_bind *)t_alloc(listen_fd, T_BIND, T_ALL))
        == NULL) {
        t_error("ERROR: t_alloc of t_bind req structure failed");
        exit(1);
    }
    if ((bindret = (struct t_bind *)t_alloc(listen_fd, T_BIND, T_ALL))
        == NULL) {
        t_error("ERROR: t_alloc of t_bind ret structure failed");
        exit(1);
    }

    /*
     * Set up t_call structure information
     */
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = INADDR_ANY;
    saddr.sin_port = htons(srv_port);
    bindreq->qlen = qlen;
    memcpy(bindreq->addr.buf, &saddr, sizeof(struct sockaddr_in));
    bindreq->addr.len = sizeof(struct sockaddr_in);
    bindret->addr.maxlen = sizeof(struct sockaddr_in);

    /*
     * Bind the endpoint to a local Internet address
     */
    if (t_bind(listen_fd, bindreq, bindret) < 0) {
        t_error("ERROR: t_bind failed for listen_fd");
        exit(1);
    }

    memcpy(&raddr, bindret->addr.buf, sizeof(struct sockaddr_in));

    /*
     * Was the correct address bound?
     */
    if (ntohs(raddr.sin_port) != srv_port) {
        sprintf(str, "t_bind bound wrong address");
        svrprint(str);
        exit(1);
    }

    /*
     * This while loop is a state machine that handles multiple connect
     * requests.  Note that signals are blocked around critical system
     * calls.
     */
    call_state = LISTEN_STATE;
    while(1) {
```

```
switch (call_state) {
case LISTEN_STATE:
    if ((call_list_entry =
         (struct call_list*)malloc(sizeof(struct call_list)))==NULL) {
        perror("ERROR: malloc of call_list_entry failed");
        exit(1);
    }
    if ((call_list_entry->call =
         (struct t_call *)t_alloc(listen_fd, T_CALL, T_ALL))==NULL) {
        t_error("ERROR: t_alloc of t_call structure failed");
        call_state = ERROR_STATE;
        break;
    }
    /*
     * link this entry to the end of the list
     */
    if (end_call == NULL) {
        end_call = first_call = call_list_entry;
    }
    else {
        end_call->next = call_list_entry;
        end_call = call_list_entry;
    }

    while(((ret = t_listen(listen_fd, call_list_entry->call)) < 0) &&
           (errno == EINTR)) {}
    if (ret < 0) {
        t_error("ERROR: t_listen failed for listen_fd");
        call_state = ERROR_STATE;
    } else {
        call_state = OPEN_RES_STATE;
    }
    break;
case OPEN_RES_STATE:
    /*
     * Since system calls can be interrupted, create a critical section
     * around these calls
     */
    sighold(SIGCLD);
    if ((end_call->resfd =
         t_open(TLI_TCP, O_RDWR,(struct t_info *)NULL)) < 0) {
        t_error("ERROR: t_open for responding fd failed");
        call_state = ERROR_STATE;
    } else {
        call_state = BIND_RES_STATE;
    }
    sigrelse(SIGCLD);
    break;
case BIND_RES_STATE:
    /*
```

```
         * Since system calls can be interrupted, create a critical sect
         * around these calls
         */
        sighold(SIGCLD);
        if (t_bind(end_call->resfd, NULL, NULL) < 0) {
            t_error("ERROR: t_bind for responding fd failed");
            call_state = ERROR_STATE;
        } else {
            call_state = ACCEPT_CALL_STATE;
        }
        sigrelse(SIGCLD);
        break;
    case ACCEPT_CALL_STATE:
        sighold(SIGCLD);
        ret = t_accept(listen_fd, first_call->resfd, first_call->call);
        sigrelse(SIGCLD);
        if (ret < 0) {
            call_state = ERROR_STATE;
            break;
        } else {
            /*
             * accept was successful
             * start child to handle call
             */
            memcpy(&calladdr, first_call->call->addr.buf,
                   sizeof(struct sockaddr_in));

            sprintf(str, "accepted call from address %s port %u",
                    inet_ntoa(calladdr.sin_addr.s_addr),
                    ntohs(calladdr.sin_port));
            svrprint(str);

            conn_fd = first_call->resfd;
            run_server(listen_fd, conn_fd, maxframe, calladdr.sin_port,
                       calladdr.sin_addr.s_addr);

            /*
             * free first entry in list and move down
             */
            tmp_call_ptr = first_call;
            if (first_call == end_call) {
                /*
                 * reached the end of the list so go back and
                 * listen for more
                 */
                first_call = end_call = NULL;
                call_state = LISTEN_STATE;
            } else {
                /*
                 * not at the end of the list so go and
```

```
                     * accept the next one
                     */
                    first_call = first_call->next;
                    call_state = ACCEPT_CALL_STATE;
                }
                t_free(tmp_call_ptr->call, T_CALL);
                free((char *)tmp_call_ptr);
            }
            break;
        case ERROR_STATE:
            if (t_errno == TLOOK) {
                process_event(listen_fd, 0, 0);
                if (t_look(listen_fd) == T_LISTEN) {
                    call_state = LISTEN_STATE;
                    break;
                }
            }
            exit(1);
            break;
        default:
            sprintf(str, "ERROR: Unknown state variable %u", call_state);
            svrprint(str);
            exit(1);
            break;
        } /* end switch */
    }
}


/*
 *  run_server spawns a new process, closes the original connection,
 *  and attempts to receive data on the new connection.  When data is
 *  received, run_server sends the data back to the client.  When an
 *  orderly release request comes from the client, run_server closes
 *  the new connection and exits.
 */

run_server(listen_fd, conn_fd, maxframe, client_port, client_addr)
    int listen_fd;
    int conn_fd;
    int maxframe;
    int client_port;
    ulong client_addr;
{
    int nbytes;
    unsigned char *buf;
    int flags;
    int look_event;
    unsigned long total_bytes = 0;
    unsigned long total_msgs = 0;
```

```
int loop_flag = 0;

switch(fork()) {
case -1:
    perror("ERROR: fork failed");
    exit(1);

default:  /* parent */
    /* close conn_fd and then go up and listen again */
    if (t_close(conn_fd) < 0) {
        t_error("ERROR: t_close failed for conn_fd");
        exit(1);
    }
    return;

case 0:  /* child */

    if ((buf = (unchar *)malloc(maxframe)) == NULL) {
        sprintf(str,"failure to malloc buf");
        svrprint(str);
        exit(1);
    }

    /* close listen_fd and do service */

    if (t_close(listen_fd) < 0) {
        t_error("ERROR: t_close failed for listen_fd");
        exit(1);
    }

    /* was disconnect already there? */
    look_event = t_look(conn_fd);
    if ((look_event != 0) && (look_event != T_DATA)) {
        sprintf(str, "t_look returned unexpected event (%d)", look_event);
        svrprint(str);
        exit(1);
    }
    /*
     * Receive a message and send it back
     */
    loop_flag = 0;
    while (!loop_flag) {
        if ((nbytes = t_rcv(conn_fd, buf, maxframe, &flags)) == -1) {
            if (t_errno == TLOOK) {
                loop_flag = process_event(conn_fd, total_msgs,
                                          total_bytes);
            }
            else {
                t_error("ERROR: t_rcv failed");
                exit(1);
```

```
                    }
                }
                else {
                    sprintf(str,"Rcvd %d bytes", nbytes);
                    svrprint(str);
                    svrprint(str);
                    if ((flags & T_EXPEDITED) > 0) {
                        sprintf(str,"T_EXPEDITED data rcvd");
                        svrprint(str);
                    }
                    total_bytes += nbytes;
                    total_msgs++;

                    if (t_snd(conn_fd, buf, nbytes, flags) < 0) {
                        t_error("ERROR: t_snd failed");
                        sprintf(str,"Exiting after %lu msgs (%lu bytes)",
                                total_msgs, total_bytes);
                        svrprint(str);
                        exit(1);
                    }
                }
            }

            if (t_close(conn_fd) < 0) {
                t_error("ERROR: t_close failed for listen_fd");
                sprintf(str,"Exiting after %lu msgs (%lu bytes)",
                        total_msgs, total_bytes);
                svrprint(str);
                exit(1);
            }
            sprintf(str, "Done");
            svrprint(str);
            exit(0);
        }
}

reapchild()
{
    int pid, status;

    do {
        pid = wait(&status);
    } while( (pid < 0) && (errno == EINTR) );
}

process_event(conn_fd, total_msgs, total_bytes)
    int conn_fd;
    unsigned int total_msgs;
    unsigned int total_bytes;
{
```

```
int look_event;
int loop_flag = 0;
static unsigned int listen_event_count=0;
static unsigned int connect_event_count=0;
static unsigned int data_event_count=0;
static unsigned int exdata_event_count=0;
static unsigned int disconnect_event_count=0;
static unsigned int error_event_count=0;
static unsigned int uderr_event_count=0;
static unsigned int ordrel_event_count=0;
struct t_discon discon;

look_event = t_look(conn_fd);
switch (look_event) {
case T_LISTEN:
    sprintf(str,"t_look returned T_LISTEN(%d)", ++listen_event_count);
    svrprint(str);
    break;
case T_CONNECT:
    sprintf(str,"t_look returned T_CONNECT(%d)", ++connect_event_count);
    svrprint(str);
    break;
case T_DATA:
    sprintf(str,"t_look returned T_DATA(%d)", ++data_event_count);
    svrprint(str);
    break;
case T_EXDATA:
    sprintf(str,"t_look returned T_EXDATA(%d)", ++exdata_event_count);
    svrprint(str);
    break;
case T_DISCONNECT:
    sprintf(str,"t_look returned T_DISCONNECT(%d)",
            ++disconnect_event_count);
    svrprint(str);
    sprintf(str,"received T_DISCONNECT after %lu msgs (%lu bytes)",
            total_msgs, total_bytes);
    svrprint(str);
    discon.udata.maxlen = 0;
    if (t_rcvdis(conn_fd, &discon) < 0) {
        t_error("ERROR: t_rcvdis failed");
    } else {
        sprintf(str,"rcvdis reason (%u) rcvdis sequence (%u)",
                discon.reason, discon.sequence);
        svrprint(str);
    }
    loop_flag++;
    break;
case T_ERROR:
    sprintf(str,"t_look returned T_ERROR(%d)", ++error_event_count);
    svrprint(str);
```

```
            loop_flag++;
            break;
        case T_UDERR:
            sprintf(str,"t_look returned T_UDERR(%d)", ++uderr_event_count);
            svrprint(str);
            loop_flag++;
            break;
        case T_ORDREL:
            sprintf(str,"t_look returned T_ORDREL(%d)", ++ordrel_event_count);
            svrprint(str);
            sprintf(str,"received T_ORDREL after %lu msgs (%lu bytes)",
                    total_msgs, total_bytes);
            svrprint(str);
            if (t_rcvrel(conn_fd) < 0) {
                t_error("ERROR: t_rcvrel failed");
                exit(1);
            }
            if (t_sndrel(conn_fd) < 0) {
                t_error("ERROR: t_sndrel failed");
                exit(1);
            }
            loop_flag++;
            break;
        default:
            sprintf(str,"t_look returned unknown event(%d)", look_event);
            svrprint(str);
            break;
    }
    return(loop_flag);
}


svrprint(str)
    char *str;
{
    time_t clock;
    char timebuf[32];

    time(&clock);
    cftime(timebuf, "%T %D", &clock);
    printf("Server: %s: %s\n", timebuf, str);
    fflush(stdout);
}
```

A.3 UDP Programming Examples

The examples in this section show a typical way of implementing a UDP client and a UDP server with the TLI.

A.3.1 UDP Client Example

```
/* $Copyright:   $
 * Copyright (c) 1984, 1985, 1986, 1987, 1988, 1989, 1990
 * Sequent Computer Systems, Inc.   All rights reserved.
 *
 * This software is furnished under a license and may be used
 * only in accordance with the terms of that license and with the
 * inclusion of the above copyright notice.   This software may not
 * be provided or otherwise made available to, or used by, any
 * other person.  No title to or ownership of the software is
 * hereby transferred.
 */

#ident "$Header: udpclient.c 1.2 90/02/13 $"
#ident "(C) 1989 Sequent Computer Systems, Inc."

/* udp_client.c */

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/tiuser.h>
#include <sys/tihdr.h>
#include <setjmp.h>
#include <stropts.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/netinet.h>

#define REMOTE_HOST "example"
#define SRV_PORT  6000
#define MAXFRAME  1048
extern int errno;


main()
{
    int tmpbytes;
    int fd;
    int i;
    int numframes = 10;
 int bytes;
 int flags = 0;
 unsigned char sendbuf[MAXFRAME];
 struct t_call *sndblk;
 struct hostent *hostptr;
 char ackbuf[MAXFRAME];
 struct t_unitdata *rcvblk;
 struct sockaddr_in saddr;
 struct sockaddr_in raddr;

 if ((fd = t_open("/dev/udp", O_RDWR, NULL)) < 0) {
t_error("Client: t_open failed");
exit(1);
 }

 /*
  * Let UDP bind to send port, don't care which one
  */

 if (t_bind(fd, NULL, NULL) < 0) {
t_error("Client: t_bind failed");
exit(1);
 }

 /*
  * put Internet address and port into sockaddr_in structure
  */

 if ((sndblk = (struct t_call *)t_alloc(fd, T_UNITDATA, T_ALL))
== NULL) {
t_error("Client: t_alloc failed");
exit(1);
 }
 if ((hostptr = gethostbyname(REMOTE_HOST)) == NULL) {
t_error("ERROR: gethostbyname failed");
exit(1);
 }

 saddr.sin_family = AF_INET;
 saddr.sin_addr.s_addr = *(unsigned long *)hostptr->h_addr;

 /*
  * Set up address to send data
  */

 saddr.sin_port = htons(SRV_PORT);
 memcpy(sndblk->addr.buf, &saddr, sizeof(struct sockaddr_in));
 memcpy(sndblk->udata.buf, sendbuf, MAXFRAME);
 sndblk->addr.len = sizeof(struct sockaddr_in);
 /*
  * Set up address/buffer to receive data
  */

 if((rcvblk=(struct t_unitdata *) t_alloc(fd, T_UNITDATA,
                  T_ALL)) == NULL) {
t_error("Client: t_alloc of t_unitdata rcvblk failed");
exit(1);
 }
 rcvblk->udata.buf = ackbuf;
 rcvblk->udata.maxlen = sizeof(ackbuf);
 rcvblk->addr.maxlen = sizeof(struct sockaddr_in);

 for (i=0; i<=10; i++) {

 /*
  * Figure random packet length and determine if it is
  * the stop packet.
  */
 bytes = lrand48() % (MAXFRAME) + 1;
 if (bytes < 4) {
     bytes = 4;
 }
 if (i == 10) {
     bytes = 4;
     sendbuf[0] = 0xff;
     sendbuf[1] = 0;
     memcpy(sndblk->udata.buf, sendbuf, bytes);
     printf("Client: Sending terminate msg, exiting\n");
 } else {
     printf("Client: Sending %d bytes\n", bytes);
 }

 /*
  * Send data
  */
 sndblk->udata.len = bytes;
 if (t_sndudata(fd, sndblk) < 0) {
     t_error("Client: t_snd failed");
     exit(1);
 }

 if (i == 10) {
     exit(0);
 }

 /*
  * Set alarm and receive data, alarm is set in case
  * data is lost in transit to or from server.  Lost
  * packets are OK in UDP
```

```
     */
    alarm(10);
    while(((tmpbytes =
       t_rcvudata(fd, rcvblk, &flags)) < 0)
          && (errno == EINTR)) {}
    alarm(0);
    if (tmpbytes < 0) {
        t_error("Client: t_rcvudata failed");
        exit(1);
    }
     }
}
```

A.3.2 UDP Server Example

```
/* $Copyright:   $
 * Copyright (c) 1984, 1985, 1986, 1987, 1988, 1989, 1990
 * Sequent Computer Systems, Inc.   All rights reserved.
 *
 * This software is furnished under a license and may be used
 * only in accordance with the terms of that license and with the
 * inclusion of the above copyright notice.   This software may not
 * be provided or otherwise made available to, or used by, any
 * other person.  No title to or ownership of the software is
 * hereby transferred.
 */

#ident "$Header: udpserver.c 1.2 90/02/13 $"
#ident "(C) 1989 Sequent Computer Systems, Inc."

/* udp_server.c */

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/tiuser.h>
#include <sys/tihdr.h>
#include <stropts.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#include <netinet/netinet.h>
#include <netinet/in.h>

extern int errno;
#define SRV_PORT 6000

main()
{
    char ackbuf[8192];
    int receive_fd;
    struct t_bind *bindreq, *bindret;
    struct t_unitdata *rcvblk, sndvblk;
    struct t_call *call;
    struct sockaddr_in saddr, raddr;
    struct sockaddr_in from;
    ulong srv_addr;
    int ret;
    int flags;

    if ((receive_fd = t_open("/dev/udp", O_RDWR, NULL)) < 0) {
   t_error("Server: t_open failed for receive_fd");
   exit(1);
    }

    /*
     * Bind to SRV_PORT
     */

    if ((bindreq = (struct t_bind *)t_alloc(receive_fd,
                    T_BIND, T_ALL)) == NULL) {
   t_error("Server: t_alloc of t_bind req structure failed");
   exit(2);
    }

    if ((bindret = (struct t_bind *)t_alloc(receive_fd,
                    T_BIND, T_ALL)) == NULL) {
   t_error("Server: t_alloc of t_bind ret structure failed");
   exit(3);
    }

    saddr.sin_addr.s_addr = INADDR_ANY;
    saddr.sin_port = htons(SRV_PORT);

    bindreq->qlen = 1;
    memcpy(bindreq->addr.buf, &saddr, sizeof(struct sockaddr_in));
    bindreq->addr.len = sizeof(struct sockaddr_in);

    bindret->addr.maxlen = sizeof(struct sockaddr_in);

    if (t_bind(receive_fd, bindreq, bindret) < 0) {
   t_error("Server: t_bind failed for receive_fd");
   exit(4);
    }

    memcpy(&raddr, bindret->addr.buf, sizeof(struct sockaddr_in));

    /*
     * Was the correct address bound
     */

    if ((raddr.sin_addr.s_addr != srv_addr) ||
    (ntohs(raddr.sin_port) != SRV_PORT)) {
    fprintf   (stderr, "Server: t_bind bound wrong address\n");
    exit(5);
     }

     /*
      * Set up address/buffer to receive data
      */

     if((rcvblk=(struct t_unitdata *) t_alloc(receive_fd, T_UNITDATA,
                        T_ALL)) == NULL) {
    t_error("Server: t_alloc of t_unitdata rcvblk failed");
    exit(6);
     }

     while(1) {
    rcvblk->udata.buf = ackbuf;
    rcvblk->udata.maxlen = sizeof(ackbuf);
    rcvblk->addr.maxlen = sizeof(struct sockaddr_in);

    /*
     * Receive packet and check for termination packet
     */

    while(((ret = t_rcvudata(receive_fd, rcvblk, &flags)) < 0)
          && (errno == EINTR)) {}
    if((unsigned char)ackbuf[0] == 0xff) {
        exit(0);
    }
    printf("Server: Got %d bytes\n", rcvblk->udata.len);
    if (ret < 0) {
        t_error("Server: t_rcvudata failed for receive_fd");
        exit(7);
    }

     /*
      * Send packet back
      */

     ret = t_sndudata(receive_fd, rcvblk, &flags);
     if (ret < 0) {
         t_error("Server: t_sndudata failed");
         exit(8);
     }
      }
}
```

Index

Index

Index

/etc/hosts file 2-2
/etc/networks file 2-5
/etc/services file 2-7
<Internet address routines
  inet_ntoa 2-14

A

Aborting a connection 3-20
Accepting connection 3-14
Address routines .... 2-9 to 10
Addresses, Internet 2-13
Allocating TLI structures 3-2

B

Berkeley Internet Name Domain Server 2-1
BIND server 2-1,2
Binding
  TCP endpoint 3-3
  UDP endpoint 4-2
Byte order
  Internet 2-11
  machine 2-11
  network 2-11
Byte order routines 2-11
  htonl 2-11
  htons 2-11
  ntohl 2-12
  ntohs 2-12

C

Closing an endpoint 3-4
Connect indication 3-15
Connect indication, listening 3-16
Connection establishment .... 3-1,14 to 17
Connection mode 1-2; 3-1

receiving data 3-18
releasing a connection 3-21
requesting a connection 3-15
sending data 3-19

Connection release .... 3-1,19 to 21
Connection, accepting 3-14
Connection, receiving 3-17
Connection, requesting 3-15
Connectionless mode 1-2

D

Data transfer, tcp 3-1
Data transfer, TCP .... 3-18 to 19
Data transfer, UDP .... 4-3 to 5
Data, receiving 3-18
Data, sending 3-19
Disconnect, receiving 3-19
Disconnect, sending 3-20
Displaying errors 3-5
Dot notation 2-13

E

endhostent 2-4
endnetent 2-6
Endpoint
  binding 3-3; 4-2
  closing 3-4
  opening 3-8
  unbinding 3-14

Endpoint address routines .... 2-9 to 10
  getmyinaddr 2-9
  getpeerinaddr 2-10

endservent 2-9
errno 1-3
Error handling 1-3
  errno 1-3
  t_errno 1-3
  t_error 1-3
  TSYSERR 1-3

Errors, displaying 3-5

Errors, UDP 4-4
Examples
  porting .... 5-13 to 25
  programming .... A-1 to 19
  TCP client A-1
  TCP server A-4
  UDP client A-14
  UDP server A-17

F-G

Freeing allocated memory 3-5
gethostbyaddr 2-3
gethostbyname 2-3
getmyinaddr 2-9
getnetbyaddr 2-6
getnetbyname 2-5
getnetent 2-6
getpeerinaddr 2-10
getservbyname 2-7
getservbyport 2-8
getservent 2-9

H

Header files 2-2; 3-1; 4-1
Host address 2-2
Host alias 2-2
Host byte order 2-11
Host name 2-2
Host name routines 2-2
  endhostent 2-4
  gethostbyaddr 2-3
  gethostbyname 2-3
  hostent structure 2-2
  sethostent 2-4
hostent structure 2-2
  host address 2-2
  host alias 2-2
  host name 2-2
Hosts database 2-1
htonl routine 2-11

htons routine 2-11

I

Include files 3-1; 4-1
inet_addr routines 2-13
inet_lnaof routines 2-14
inet_makeaddr routines 2-13
inet_netof routines 2-14
inet_network routines 2-13
inet_ntoa routines 2-14
Initiating a connection 3-15
Internet address routines 2-13
  inet_addr 2-13
  inet_lnaof 2-14
  inet_makeaddr 2-13
  inet_netof 2-14
  inet_network 2-13
Internet addresses 2-13
  dot notation 2-13
Internet byte order 2-11

J-K-L

Libraries 3-1
Linger time 3-11
Listening for connection indication 3-16
Local management .... 3-1 to 14; 4-1 to 3

M

Machine byte order 2-11
Memory, allocating 3-2
Memory, freeing 3-5

N

netent structure 2-5
  network alias 2-5
  network name 2-5
  network type 2-5
Network alias 2-5

Network byte order 2-11
Network database 2-1,5
Network name 2-5
Network name routines 2-5
  endnetent 2-6
  getnetbyaddr 2-6
  getnetbyname 2-5
  getnetent 2-6
  netent structure 2-5
  setnetent 2-6
Network type 2-5
ntohl routine 2-12
ntohs 2-12

O

Opening a TCP endpoint 3-8
Opening a UDP endpoint 4-2
Options .... 3-9 to 12
Options negotiation 3-9,12
Orderly release, receiving 3-20
Orderly release, sending 3-21
Outstanding connect indications 3-15

P

Porting
  client addresses 5-11
  connect requests 5-8
  connection refusal 5-11
  examples .... 5-13 to 25
  read 5-11
  signals 5-1
  TCP client 5-9
  TCP server 5-5
  tirdwr module 5-11
  UDP programs 5-12
  write 5-11
Porting sockets to TLI .... 5-1 to 25
Programming examples .... A-1 to 19

Q-R

Receiving a connection 3-17
Receiving a release 3-20
Receiving data, TCP 3-18
Receiving data, UDP 4-4
Receiving disconnects 3-19
Rejecting a connect request 3-20
Release, receiving 3-20
Release, sending 3-21
Releasing a connection 3-21
Requesting a connection 3-15
Returning state information 3-7

S

Sending a disconnect 3-20
Sending a release 3-21
Sending data, TCP 3-19
Sending data, UDP 4-5
servent structure 2-7
  service address 2-7
  service alias 2-7
  service name 2-7
Service alias 2-7
Service modes 1-2
  connection 1-2
  connectionless 1-2
Service name 2-7
Service name routines 2-7
  endservent 2-9
  getservbyname 2-7
  getservbyport 2-8
  getservent 2-9
  servent structure 2-7
  setservent 2-9
Service port 2-7
sethostent 2-4
setnetent 2-6
setservent 2-9
Signals 5-1
Standard byte order 2-11

State information 3-7
STREAMS 1-1
Synchronizing library structures 3-13

T

t_accept 3-14
t_alloc 3-2
t_bind 3-3; 4-2
t_close 3-4
t_connect 3-15
t_errno 1-3
t_error 1-3; 3-5
t_free 3-5
t_getinfo 3-6
t_getstate 3-7
t_listen 3-16
t_look 3-7
t_open 3-8; 4-2
t_optmgmt 3-9
t_rcv 3-18
t_rcvconnect 3-17
t_rcvdis 3-19
t_rcvrel 3-20
t_rcvudata 4-4
t_rcvuderr 4-4
t_snd 3-19
t_snddis 3-20
t_sndrel 3-21
t_sndudata 4-5
t_sync 3-13
t_unbind 3-14
TCP 1-2
TCP phases
  connection establishment 3-1
  connection release 3-1
  data transfer 3-1
  local management 3-1
TCP_SETOPT 3-12
TLI 1-1
  error handling 1-3
TLI structures 3-2

Transmission Control Protocol 1-2
Transport Level Interface 1-1
TSYSERR 1-3

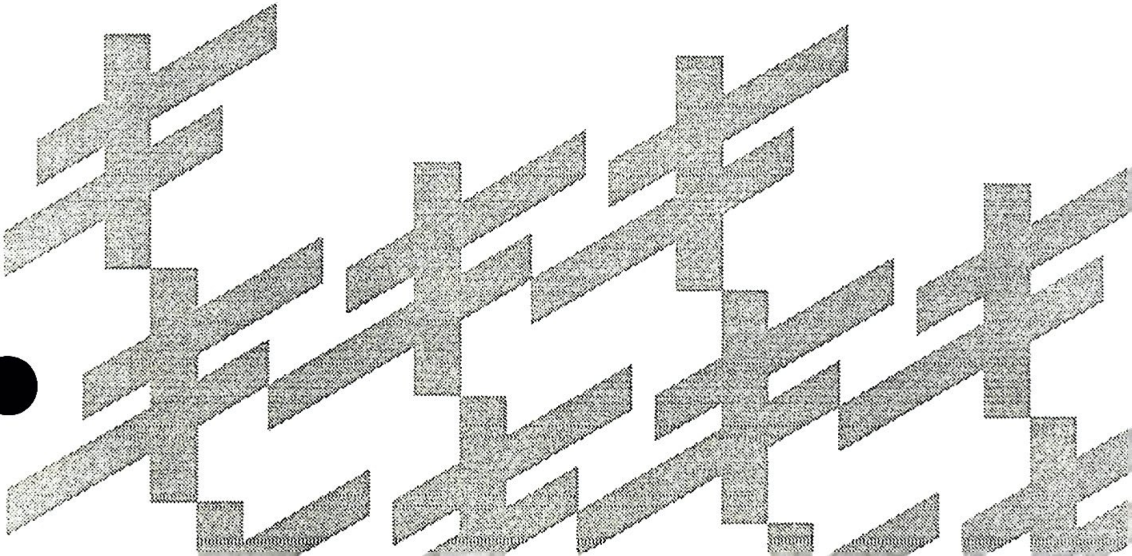
U-V-W-X-Y-Z

UDP .... 1-2; 4-1 to 5
UDP errors 4-4
Unbinding endpoint 3-14
User Datagram Protocol 1-2; 4-1

ptx/TCP/IP Reference Manual

ptx/TCP/IP™
Reference Manual

1003-48920-00
Sequent Computer Systems, Inc.



DYNIX/ptx and ptx/TCP/IP are trademarks, and Symmetry is a registered trademark of Sequent Computer Systems, Inc.

Ethernet is a trademark of Xerox Corporation.

PDP is a trademark of Digital Equipment Corporation.

UNIX is a registered trademark of AT&T.

Copyright© 1990 by Sequent Computer Systems, Inc. All rights reserved. This document may not be copied or reproduced in any form without permission from Sequent Computer Systems, Inc. Information in this document is subject to change without notice.

Printed in the United States of America.

Permuted Index

/returns current IP and port address for local endpoint....getmyinaddr(3N)
/returns current IP and port address for remote endpoint....getpeerinaddr(3N)
/inet_lnaof, inet_netof: Internet address manipulation routines....inet(3N)
arp: address resolution display and control....arp(1C)
control arp: address resolution display and....arp(1C)
ftp: ARPANET file transfer program....ftp(1C)
syslog, openlog, closelog, setlogmask: control system log....syslog(3)
talkd: remote user communication server....talkd(1M)
resolver: resolver configuration file....resolver(5)
network/ htonl, htons, ntohl, ntohs: convert values between host and....byteorder(3N)
rcp: remote file copy....rcp(1C)
netd: network daemon....netd(1M)
ftpd: DARPA Internet File Transfer Protocol server....ftpd(1M)
telnetd: DARPA TELNET protocol server....telnetd(1M)
tftpd: DARPA Trivial File Transfer Protocol server....tftpd(1M)
hosts: host name data base....hosts(5)
networks: network name data base....networks(5)
services: service name data base....services(5)
shells: shell database....shells(5)
udp: User Datagram Protocol....udp(7)
arp: address resolution display and control....arp(1C)
res_mkquery, res_send, res_init, dn_comp, dn_expand: resolver/....resolver(3N)
/res_send, res_init, dn_comp, dn_expand: resolver routines....resolver(3N)
named: Internet domain name server....named(1M)
Control Protocol STREAMS driver....tcp(7)
rdump: file system dump across the network....rdump(1C)
rrestore: restore a file system dump across the network....rrestore(1C)
ping: send ICMP ECHO_REQUEST packets to network hosts....ping(1C)
/gethostbyname, sethostent, endhostent: get network host entry....gethostbyname(3N)
/getnetbyname, setnetent, endnetent: get network entry....getnetent(3N)
IP and port address for local endpoint....getmyinaddr(3N)
IP and port address for remote endpoint....getpeerinaddr(3N)
/getservbyname, setservent, endservent: get service entry....getservent(3N)
rexecd: remote execution server....rexecd(1M)
resolver: resolver configuration file....resolver(5)
rcp: remote file copy....rcp(1C)
rdump: file system dump across the network....rdump(1C)
rrestore: restore a file system dump across the network....rrestore(1C)
ftp: ARPANET file transfer program....ftp(1C)
tftp: trivial file transfer program....tftp(1C)
ftpd: DARPA Internet File Transfer Protocol server....ftpd(1M)
tftpd: DARPA Trivial File Transfer Protocol server....tftpd(1M)
ftp: ARPANET file transfer program....ftp(1C)
Protocol server ftpd: DARPA Internet File Transfer....ftpd(1M)
sethostent, endhostent: get network/ gethostbyaddr, gethostbyname,....gethostbyname(3N)
endhostent: get/ gethostbyaddr, gethostbyname, sethostent,....gethostbyname(3N)
port address for local endpoint getmyinaddr: returns current IP and....getmyinaddr(3N)

setnetent, endnetent:/ getnetent,	getnetbyaddr, getnetbyname,	getnetent(3N)
getnetent, getnetbyaddr,	getnetbyname, setnetent, endnetent: get/	getnetent(3N)
getnetbyname, setnetent, endnetent:/	getnetent, getnetbyaddr,	getnetent(3N)
and port address for remote/	getpeerinaddr: returns current IP	getpeerinaddr(3N)
getservent, getservbyport,	getservbyname, setservent,/	getservent(3N)
setservent, endservent:/ getservent,	getservbyport, getservbyname,	getservent(3N)
getservbyname, setservent,/	getservent, getservbyport,	getservent(3N)
ntohl, ntohs: convert values between	host and network byte order	byteorder(3N)
sethostent, endhostent: get network	host entry	gethostbyname(3N)
hosts:	host name data base	hosts(5)
ruptime: show	host status of local machines	ruptime(1C)
hosts.equiv: list of trusted	hosts	hosts.equiv(5)
ICMP ECHO_REQUEST packets to network	hosts	ping(1C)
	hosts: host name data base	hosts(5)
	hosts.equiv: list of trusted hosts	hosts.equiv(5)
values between host and network/	htonl, htons, ntohl, ntohs: convert	byteorder(3N)
between host and network/ htonl,	htons, ntohl, ntohs: convert values	byteorder(3N)
hosts ping: send	ICMP ECHO_REQUEST packets to network	ping(1C)
inet_makeaddr, inet_lnaof,/	inet_addr, inet_network, inet_ntoa,	inet(3N)
	inetd: internet super–server	inetd(1M)
/inet_ntoa, inet_makeaddr,	inet_lnaof, inet_netof: Internet address/	inet(3N)
inet_addr, inet_network, inet_ntoa,	inet_makeaddr, inet_lnaof,/	inet(3N)
/inet_makeaddr, inet_lnaof,	inet_netof: Internet address/	inet(3N)
inet_addr,	inet_network, inet_ntoa, inet_makeaddr,/	inet(3N)
inet_addr, inet_network,	inet_ntoa, inet_makeaddr,/	inet(3N)
nslookup: query name servers	interactively	nslookup(1C)
telnet: user	interface to the TELNET protocol	telnet(1C)
sendmail: send mail over the	internet	sendmail(1M)
/inet_lnaof, inet_netof:	Internet address manipulation routines	inet(3N)
named:	Internet domain name server	named(1M)
ftpd: DARPA	Internet File Transfer Protocol server	ftpd(1M)
ip:	Internet Protocol	ip(7)
inetd:	internet super–server	inetd(1M)
Protocol STREAMS driver TCP:	Internet Transmission Control	tcp(7)
getmyinaddr: returns current	IP and port address for local/	getmyinaddr(3N)
getpeerinaddr: returns current	IP and port address for remote/	getpeerinaddr(3N)
	ip: Internet Protocol	ip(7)
hosts.equiv:	list of trusted hosts	hosts.equiv(5)
closelog, setlogmask: control system	log	syslog(3)
syslogd:	log systems messages	syslogd(1M)
rwho: whos	logged in on local machines	rwho(1C)
rlogin: remote	login	rlogin(1C)
rlogind: remote	login server	rlogind(1M)
ruptime: show host status of local	machines	ruptime(1C)
rwho: whos logged in on local	machines	rwho(1C)
rmt: remote	magtape protocol module	rmt(1C)
sendmail: send	mail over the internet	sendmail(1M)
inet_netof: Internet address	manipulation routines	inet(3N)
route:	manually manipulate the routing tables	route(1C)
syslogd: log systems	messages	syslogd(1M)
rlmod: rlogin protocol	module	rlmod(1M)
rmt: remote magtape protocol	module	rmt(1C)

tnmod: telnet protocol	module	tnmod(1M)
hosts: host	name data base	hosts(5)
networks: network	name data base	networks(5)
services: service	name data base	services(5)
named: Internet domain	name server	named(1M)
nslookup: query	name servers interactively	nslookup(1C)
	named: Internet domain name server	named(1M)
	netd: network daemon	netd(1M)
	netstat: show network status	netstat(1A)
rdump: file system dump across the	network	rdump(1C)
a file system dump across the	network	rrestore(1C)
convert values between host and	network byte order	byteorder(3N)
netd:	network daemon	netd(1M)
setnetent, endnetent: get	network entry	getnetent(3N)
sethostent, endhostent: get	network host entry	gethostbyname(3N)
send ICMP ECHO_REQUEST packets to	network hosts	ping(1C)
networks:	network name data base	networks(5)
netstat: show	network status	netstat(1A)
	networks: network name data base	networks(5)
interactively	nslookup: query name servers	nslookup(1C)
host and network byte/ htonl, htons,	ntohl, ntohs: convert values between	byteorder(3N)
and network/ htonl, htons, ntohl,	ntohs: convert values between host	byteorder(3N)
control system log syslog,	openlog, closelog, setlogmask:	syslog(3)
values between host and network byte	order	byteorder(3N)
ping: send ICMP ECHO_REQUEST	packets to network hosts	ping(1C)
to network hosts	ping: send ICMP ECHO_REQUEST packets	ping(1C)
getmyinaddr: returns current IP and	port address for local endpoint	getmyinaddr(3N)
/returns current IP and	port address for remote endpoint	getpeerinaddr(3N)
ftp: ARPANET file transfer	program	ftp(1C)
tftp: trivial file transfer	program	tftp(1C)
ip: Internet	Protocol	ip(7)
telnet: user interface to the TELNET	protocol	telnet(1C)
udp: User Datagram	Protocol	udp(7)
rlmod: rlogin	protocol module	rlmod(1M)
rmt: remote magtape	protocol module	rmt(1C)
tnmod: telnet	protocol module	tnmod(1M)
ftpd: DARPA Internet File Transfer	Protocol server	ftpd(1M)
telnetd: DARPA TELNET	protocol server	telnetd(1M)
tftpd: DARPA Trivial File Transfer	Protocol server	tftpd(1M)
TCP: Internet Transmission Control	Protocol STREAMS driver	tcp(7)
nslookup:	query name servers interactively	nslookup(1C)
for returning a stream to a remote/	rcmd, rresvport, ruserok: routines	rcmd(3X)
	rcp: remote file copy	rcp(1C)
network	rdump: file system dump across the	rdump(1C)
routines for returning a stream to a	remote command	rcmd(3X)
rexec: return stream to a	remote command	rexec(3X)
current IP and port address for	remote endpoint	getpeerinaddr(3N)
rexecd:	remote execution server	rexecd(1M)
rcp:	remote file copy	rcp(1C)
rlogin:	remote login	rlogin(1C)
rlogind:	remote login server	rlogind(1M)
rmt:	remote magtape protocol module	rmt(1C)

resh:	remote shell	resh(1C)
reshd:	remote shell server	reshd(1M)
talkd:	remote user communication server	talkd(1M)
	resh: remote shell	resh(1C)
	reshd: remote shell server	reshd(1M)
res_mkquery, res_send,	res_init, dn_comp, dn_expand: resolver/	resolver(3N)
dn_comp, dn_expand: resolver/	res_mkquery, res_send, res_init,	resolver(3N)
arp: address	resolution display and control	arp(1C)
resolver:	resolver configuration file	resolver(5)
file	resolver: resolver configuration	resolver(5)
res_init, dn_comp, dn_expand:	resolver routines	resolver(3N)
dn_expand: resolver/ res_mkquery,	res_send, res_init, dn_comp,	resolver(3N)
the network rrestore:	restore a file system dump across	rrestore(1C)
rexec:	return stream to a remote command	rexec(3X)
/rresvport, ruserok: routines for	returning a stream to a remote/	rcmd(3X)
for local endpoint getmyinaddr:	returns current IP and port address	getmyinaddr(3N)
for remote endpoint getpeerinaddr:	returns current IP and port address	getpeerinaddr(3N)
command	rexec: return stream to a remote	rexec(3X)
	rexecd: remote execution server	rexecd(1M)
	rlmod: rlogin protocol module	rlmod(1M)
rlmod:	rlogin protocol module	rlmod(1M)
	rlogin: remote login	rlogin(1C)
	rlogind: remote login server	rlogind(1M)
	rmt: remote magtape protocol module	rmt(1C)
routing tables	route: manually manipulate the	route(1C)
Internet address manipulation	routines	inet(3N)
dn_comp, dn_expand: resolver	routines	resolver(3N)
rcmd, rresvport, ruserok:	routines for returning a stream to a remote/	rcmd(3X)
route: manually manipulate the	routing tables	route(1C)
across the network	rrestore: restore a file system dump	rrestore(1C)
returning a stream to a/ rcmd,	rresvport, ruserok: routines for	rcmd(3X)
machines	ruptime: show host status of local	ruptime(1C)
stream to a remote/ rcmd, rresvport,	ruserok: routines for returning a	rcmd(3X)
machines	rwho: whos logged in on local	rwho(1C)
	rwhod: system status server	rwhod(1M)
network hosts ping:	send ICMP ECHO_REQUEST packets to	ping(1C)
sendmail:	send mail over the internet	sendmail(1M)
internet	sendmail: send mail over the	sendmail(1M)
Internet File Transfer Protocol	server	ftpd(1M)
inetd: internet super–	server	inetd(1M)
named: Internet domain name	server	named(1M)
reshd: remote shell	server	reshd(1M)
rexecd: remote execution	server	rexecd(1M)
rlogind: remote login	server	rlogind(1M)
rwhod: system status	server	rwhod(1M)
talkd: remote user communication	server	talkd(1M)
telnetd: DARPA TELNET protocol	server	telnetd(1M)
DARPA Trivial File Transfer Protocol	server	tftpd(1M)
nslookup: query name	servers interactively	nslookup(1C)
	services: service name data base	services(5)
host/ gethostbyaddr, gethostbyname,	sethostent, endhostent: get network	gethostbyname(3N)
syslog, openlog, closelog,	setlogmask: control system log	syslog(3)

/getnetbyaddr, getnetbyname,	setnetent, endnetent: get network entry	getnetent(3N)
entry/getservbyport, getservbyname,	setservent, endservent: get service	getservent(3N)
resh: remote	shell	resh(1C)
shells:	shell database	shells(5)
reshd: remote	shell server	reshd(1M)
	shells: shell database	shells(5)
netstat: show network	status	netstat(1A)
ruptime: show host	status of local machines	ruptime(1C)
rwhod: system	status server	rwhod(1M)
/ruserok: routines for returning a	stream to a remote command	rcmd(3X)
rexec: return	stream to a remote command	rexec(3X)
Transmission Control Protocol	STREAMS driver	tcp(7)
inetd: internet	super–server	inetd(1M)
setlogmask: control system log	syslog, openlog, closelog,	syslog(3)
	syslogd: log systems messages	syslogd(1M)
syslogd: log	systems messages	syslogd(1M)
manually manipulate the routing	tables	route(1C)
	talk: talk to another user	talk(1C)
talk:	talk to another user	talk(1C)
server	talkd: remote user communication	talkd(1M)
Protocol STREAMS driver	TCP: Internet Transmission Control	tcp(7)
telnet: user interface to the	TELNET protocol	telnet(1C)
tnmod:	telnet protocol module	tnmod(1M)
telnetd: DARPA	TELNET protocol server	telnetd(1M)
protocol	telnet: user interface to the TELNET	telnet(1C)
server	telnetd: DARPA TELNET protocol	telnetd(1M)
	tftp: trivial file transfer program	tftp(1C)
Protocol server	tftpd: DARPA Trivial File Transfer	tftpd(1M)
	tnmod: telnet protocol module	tnmod(1M)
ftp: ARPANET file	transfer program	ftp(1C)
tftp: trivial file	transfer program	tftp(1C)
ftpd: DARPA Internet File	Transfer Protocol server	ftpd(1M)
tftpd: DARPA Trivial File	Transfer Protocol server	tftpd(1M)
TCP: Internet	Transmission Control Protocol STREAMS driver	tcp(7)
tftp:	trivial file transfer program	tftp(1C)
tftpd: DARPA	Trivial File Transfer Protocol server	tftpd(1M)
hosts.equiv: list of	trusted hosts	hosts.equiv(5)
	udp: User Datagram Protocol	udp(7)
talk: talk to another	user	talk(1C)
talkd: remote	user communication server	talkd(1M)
udp:	User Datagram Protocol	udp(7)
telnet:	user interface to the TELNET protocol	telnet(1C)
htonl, htons, ntohl, ntohs: convert	values between host and network byte/	byteorder(3N)
rwho:	whos logged in on local machines	rwho(1C)

NAME

netstat – show network status

SYNOPSIS

netstat [**–an**] [**–a**]
netstat [**–imnrs**]

DESCRIPTION

netstat symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of protocol control blocks for each protocol. The second form presents the contents of one of the other network data structures according to the option selected.

The options have the following meaning:

–a	show the state of all protocol control blocks; normally protocol control blocks used by server processes are not shown.
–i	show the state of interfaces which have been auto-configured; interfaces statically configured into a system, but not located at boot time are not shown.
–m	show statistics recorded by the memory management routines and the relevant streams buffer information.
–n	show network addresses as numbers (normally **netstat** interprets addresses and attempts to display them symbolically).
–s	show per-protocol statistics.
–r	show the routing tables. When **–s** is also present, show routing statistics instead.

There are a number of display formats, depending on the information presented. The default display, for active protocol control blocks, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol.

Address formats are of the form "host.port" or "network.port" if a protocol control block's address specifies a network but no specific host address. When known, the host and network addresses are displayed symbolically according to the databases */etc/hosts* and */etc/networks*, respectively, or through the name server. If a symbolic name for an address is unknown, or if the **–n** option is specified, the address is printed in the Internet "dot format". Unspecified, or "wildcard", addresses and ports appear as "*".

The interface display provides a table of cumulative statistics regarding packets transferred and packets dropped on output. The network address (currently Internet specific) of the interface is also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route ("U" if "up"), whether the route is to a gateway ("G"), whether the route was created dynamically by a redirect ("D"), whether the route has been dynamically modified ("M"), and whether the route is permanent ("P").

Direct routes are created for each interface attached to the local host. The TT field indicates the remaining time-to-live for this route. PERM routes are created via the route command and are never timed out. A route begins aging after each reference. Each reference to the route resets the timer to the maximum value. The *use* field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

SEE ALSO

route(1C), netd(1C)

NAME

arp – address resolution display and control

SYNOPSIS

arp *hostname*
arp –a
arp –d *hostname*
arp –s *hostname ether_addr* [**temp**] [**pub**]
arp –f *filename*

DESCRIPTION

The **arp** program displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol.

With no flags, the program displays the current ARP entry for *hostname.* The host may be specified by name or by number, using Internet dot notation. With the **–a** flag, the program displays all of the current ARP entries.

With the **–d** flag, a superuser may delete an entry for the host called *hostname.*

The **–s** flag is given to create an ARP entry for the host called *hostname* with the Ethernet address *ether_addr.* The Ethernet address is given as six hex bytes separated by colons. The entry will be permanent unless **temp** is specified in the command. If **pub** is used, the entry will be published, meaning this system will act as an ARP server, responding to requests for *hostname* even though the host address is not its own.

The **–f** flag causes the file *filename* to be read and multiple entries to be set in the ARP tables. Entries in the file should be of the form:

hostname ether_addr [**temp**] [**pub**]

SEE ALSO

inetd(1M), arp(1C)

NAME

ftp – ARPANET file transfer program

SYNOPSIS

ftp [–v] [–d] [–i] [–n] [–g] [host]

DESCRIPTION

ftp is the user interface to the ARPANET standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which **ftp** is to communicate may be specified on the command line. If this is done, **ftp** immediately attempts to establish a connection to an FTP server on that host; otherwise, **ftp** enters its command interpreter and awaits instructions from the user. When **ftp** is awaiting commands from the user, the prompt "ftp>" is displayed. The following commands are recognized by **ftp**:

! [command [*args*]]
Invoke an interactive shell on the local machine. If there are arguments, the first is executed directly as a command, and the rest are interpreted as arguments for that command.

$ *macro-name* [*args*]
Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro unglobbed.

account [*passwd*]
Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user is prompted for an account password in a nonechoing input mode.

append *local-file* [*remote-file*]
Append a local file to a file on the remote machine. If *remote-file* is unspecified, the local file name is used to name the remote file after being altered by any **ntrans** or **nmap** setting. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

ascii Set the file transfer **type** to network ASCII. This is the default type.

bell Arrange that a bell be sounded after each file transfer command is completed.

binary
Set the file transfer **type** to support binary image transfer.

bye Terminate the FTP session with the remote server and exit **ftp**. An end-of-file also terminates the session and exits.

case Toggle remote computer file name case mapping during **mget** commands. When **case** is on (default is off), remote computer file names with all letters in uppercase are written in the local directory with the letters mapped to lowercase.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

cdup Change the remote machine working directory to the parent of the current remote machine working directory.

close Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

cr Toggle carriage return stripping during ascii type file retrieval. Records are denoted by a carriage return/linefeed sequence during ASCII-type file transfer. When **cr** is on (the default), carriage returns are stripped from this sequence to conform with the UNIX single linefeed record delimiter. Records on non-UNIX remote systems may contain single linefeeds; when an ASCII-type transfer is made, these linefeeds may be distinguished from a record delimiter only when **cr** is off.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified, it is used to set the debugging level. When debugging is on, **ftp** prints each command sent to the remote machine, preceded by the string "-->".

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, place the output in *local-file*. If interactive prompting is on, **ftp** prompts the user to verify that the last argument is indeed the target local file for receiving **dir** output. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is –, is sent to the terminal.

disconnect
A synonym for **close**.

form *format*
Set the file transfer **form** to *format*. The default format is "file."

get *remote-file* [*local-file*]
Retrieve the *remote-file*, and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current **case**, **ntrans**, and **nmap** settings. The current settings for *type*, *form*, *mode*, and *structure* are used while transferring the file.

glob Toggle filename expansion for **mdelete**, **mget** and **mput**. If globbing is turned off with **glob**, the file name arguments are taken literally and are not expanded. Globbing for **mput** is done as in **csh**(1). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine, and the lists are not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file: the exact result depends on the foreign operating system and **ftp** server, and can be previewed by issuing "**mls** *remote-files* -". Note: **mget** and **mput** are not meant to transfer entire directory subtrees of files. That can be done by transferring a **tar**(1) archive of the subtree (in binary mode).

hash Toggle hash-sign ("#") printing for each data block transferred. The size of a data block is 1024 bytes.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, **ftp** prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is unspecified, the current working directory is used. If interactive prompting is on, **ftp** prompts the user to verify that the last argument is indeed the target local file for receiving **ls** output. If no local file is specified, or if *local-file* is -, the output is sent to the terminal.

macdef *macro-name*
Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed. The macro processor interprets "$" and "\" as special characters. A "$" followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A "$" followed by an "i" signals that macro processor that the executing macro is to be looped. On the first pass "$i" is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A "\" followed by any character is replaced by that character. Use the "\" to prevent special treatment of the "$".

mdelete [*remote-files*]
Delete the *remote-files* on the remote machine.

mdir *remote-files local-file*
Like **dir**, except multiple *remote-files* may be specified. If interactive prompting is on, **ftp** prompts the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

mget *remote-files*
Expand the *remote-files* on the remote machine, and do a get for each file name thus produced. See **glob** for details on the filename expansion. Resulting file names will then be processed according to **case**, **ntrans**, and **nmap** settings. Files are transferred into the local working directory, which can be changed with an **lcd** *directory* command; new local directories can be created with a **! mkdir** *directory* command.

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Like **ls**, except multiple *remote-files* may be specified. If interactive prompting is on, **ftp** prompts the user to verify that the last argument

is indeed the target local file for receiving **mls** output.

mode [*mode-name*]

Set the file transfer *mode* to *mode-name*. The default mode is "stream" mode.

mput *local-files*

Expand wildcards in the list of local files given as arguments, and do a **put** for each file in the resulting list. See **glob** for details of filename expansion. Resulting file names will then be processed according to **ntrans** and **nmap** settings.

nmap [*inpattern outpattern*]

Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during **mput** commands, and **put** commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during **mget** commands, and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file-naming conventions or practices.

The mapping follows the pattern set by *inpattern* and *outpattern*. *inpattern* is a template for incoming filenames (which may have already been processed according to the **ntrans** and **case** settings). Variable templating is accomplished by including the sequences "$1", "$2", ..., "$9" in *inpattern*. Use "\" to prevent this special treatment of the "$" character. All other characters are treated literally and are used to determine the **nmap** *inpattern* variable values.

For example, given *inpattern* $1.$2 and the remote file name "*mydata.data*", $1 would have the value "mydata", and $2 would have the value "data". The *outpattern* determines the resulting mapped filename. The sequences "$1", "$2", ..., "$9" are replaced by any value resulting from the *inpattern* template. The sequence "$0" is replace by the original filename. Additionally, the sequence "[*seq1*,*seq2*]" is replaced by *seq1* if *seq1* is not a null string; otherwise, it is replaced by *seq2*. For example, the command "**nmap** $1.$2.$3 [$1,$2].[$2,file]" would yield the output filename *myfile.data* for input filenames *myfile.data* and *myfile.data.old*, *myfile.file* for the input filename *myfile*, and *myfile.myfile* for the input filename *.myfile*.

Spaces may be included in *outpattern*, as in the example: nmap $1 | sed "s/ *$//" > $1 . Use the "\" character to prevent special treatment of the "$", "[", "]", and "," characters.

ntrans [*inchars* [*outchars*]]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during **mput** commands, and **put** commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during **mget** commands, and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file-naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

open *host* [*port*]

Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, **ftp** attempts to contact an FTP server at that port. If the *auto-login* option is on (default), **ftp** also attempts to automatically log the user in to the FTP server (see below).

prompt

Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default is on), any **mget** or **mput** commands will transfer all files, and any **mdelete** commands will delete all files.

proxy *ftp-command*

Execute an ftp command on a secondary control connection. This command allows simultaneous connection to two remote ftp servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command "proxy ?" to see other ftp commands executable on the secondary connection. The following commands behave differently when prefaced by **proxy**: **open** will not define new macros during the auto-login process, **close** will not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mput**, and **append** transfer files from the host on the secondary control connection to the host on the primary control connection. Third-party file transfers depend upon support of the ftp protocol PASV command by the server on the secondary control connection.

put *local-file* [*remote-file*]

Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any **ntrans** or **nmap** settings in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for **bye**.

quote *arg1 arg2 ...*

The arguments specified are sent, verbatim, to the remote FTP server.

recv *remote-file* [*local-file*]

A synonym for **get**.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified, it is supplied to the server as well.

rename [*from*] [*to*]
Rename the file *from* on the remote machine, to the file *to*.

reset Clear reply queue. This command resynchronizes command/reply sequencing with the remote ftp server. Resynchronization may be necessary following a violation of the ftp protocol by the remote server.

rmdir *directory-name*
Delete a directory on the remote machine.

runique
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a **get** or **mget** command, a ".1" is appended to the name. If the resulting name matches another existing file, a ".2" is appended to the original name. If this process continues up to ".99", an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note that **runique** will not affect local files generated from a shell command (see below). The default value is off.

send *local-file* [*remote-file*]
A synonym for **put**.

sendport
Toggle the use of PORT commands. By default, **ftp** attempts to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, **ftp** uses the default data port. When the use of PORT commands is disabled, no attempt is made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they've been accepted.

status Show the current status of **ftp**.

struct [*struct-name*]
Set the file transfer structure to *struct-name*. By default "stream" structure is used.

sunique
Toggle storing of files on remote machine under unique file names. Remote ftp server must support ftp protocol STOU command for successful completion. The remote server will report unique name. Default value is off.

tenex Set the file transfer type to that needed to talk to TENEX machines.

trace Toggle packet tracing.

type [*type-name*]
Set the file transfer **type** to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not

specified and the server requires it, **ftp** prompts the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user is prompted for it. If an account field is specified, an account command is relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless **ftp** is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

verbose

Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]

A synonym for **help**.

Command arguments which have embedded spaces may be quoted with quote (") marks.

ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key (usually Ctrl-C). Sending transfers will be immediately halted. Receiving transfers will be halted by sending an ftp protocol ABOR command to the remote server and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an "ftp>" prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when **ftp** has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local **ftp** program must be killed by hand.

FILE NAMING CONVENTIONS

Files specified as arguments to **ftp** commands are processed according to the following rules.

1) If the file name "–" is specified, the **stdin** (for reading) or **stdout** (for writing) is used.

2) If the first character of the file name is "|," the remainder of the argument is interpreted as a shell command. **ftp** then forks a shell, using *popen*(3) with the argument supplied, and reads (writes) from the **stdout** (**stdin**). If the shell command includes spaces, the argument must be quoted; e.g. "| ls -lt". A particularly useful example of this mechanism is "dir |more."

3) Failing the above checks, if "globbing" is enabled, local file names are expanded according to the rules used in the **csh**(1); c.f. the **glob** command. If the **ftp** command expects a single local file (.e.g. **put**), only the first filename generated by the "globbing" operation is used.

4) For **mget** commands and **get** commands with unspecified local file names, the local filename is the remote filename, which may be

altered by a **case**, **ntrans**, or **nmap** setting. The resulting filename may then be altered if **runique** is on.

5) For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a **ntrans** or **nmap** setting. The resulting filename may then be altered by the remote server if **sunique** is on.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The **type** may be one of "ascii", "image" (binary), "ebcdic", and "local byte size" (for PDP-10's and PDP-20's mostly). **ftp** supports the ASCII and image types of file transfer, plus local byte size 8 for **tenex** mode transfers.

ftp supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

OPTIONS

Options may be specified at the command line or to the command interpreter.

The **–v** (verbose on) option forces **ftp** to show all responses from the remote server, as well as report on data transfer statistics.

The **–n** option restrains **ftp** from attempting "auto-login" upon initial connection. If auto-login is enabled, **ftp** checks the *.netrc* (see below) file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, **ftp** prompts for the remote machine login name (default is the user identity on the local machine), and, if necessary, prompts for a password and an account with which to log in.

The **–i** option turns off interactive prompting during multiple file transfers.

The **–d** option enables debugging.

The **–g** option disables file name globbing.

THE .netrc FILE

The *.netrc* file contains login and initialization information used by the auto-login process. It resides in the user's home directory. The following tokens are recognized; they may be separated by spaces, tabs, or new-lines:

machine *name*

Identify a remote machine name. The auto-login process searches the *.netrc* file for a **machine** token that matches the remote machine specified on the **ftp** command line or as an **open** command argument. Once a match is made, the subsequent *.netrc* tokens are processed, stopping when the end-of-file is reached or another **machine** token is encountered.

login *name*

Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

password *string*

Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the *.netrc* file, **ftp** will abort the auto-login process if the *.netrc* is readable by anyone besides the user.

account *string*

Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password; the auto-login process will initiate an ACCT command if it does not.

macdef *name*

Define a macro. This token functions like the **ftp macdef** command functions. A macro is defined with the specified name; its contents begin with the next *.netrc* line and continue until a null line (consecutive new-line characters) is encountered. If a macro named **init** is defined, it is automatically executed as the last step in the auto-login process.

SEE ALSO

ftpd(1M)

BUGS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2 BSD UNIX ASCII-mode transfer code has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2 BSD servers using the ASCII type. Avoid this problem by using the binary image type.

NAME

nslookup – query name servers interactively

SYNOPSIS

nslookup [*host-to-find* | – [*server address* | *server name*]]

DESCRIPTION

nslookup is a program to query DARPA Internet domain name servers. **nslookup** has two modes: interactive and noninteractive. Interactive mode allows the user to query the name server for information about various hosts and domains or print a list of hosts in the domain. Noninteractive mode is used to print just the name and Internet address of a host or domain.

ARGUMENTS

Interactive mode is entered in the following cases:

a) when no arguments are given (the default name server will be used), and

b) when the first argument is a hyphen (–) and the second argument is the host name of a name server.

Noninteractive mode is used when the name of the host to be looked up is given as the first argument. The optional second argument specifies a name server.

INTERACTIVE COMMANDS

Commands may be interrupted at any time by typing a Crtl-C. To exit, type a Cntl-D (EOF). The command line length must be less than 80 characters. An unrecognized command will be interpreted as a host name.

host [*server*]

Look up information for *host* using the current default server or using *server* if it is specified.

server *domain*
lserver *domain*

Change the default server to *domain*. **Lserver** uses the initial server to look up information about *domain*, while **server** uses the current default server. If an authoritative answer can't be found, the names of servers that might have the answer are returned.

root Changes the default server to the server for the root of the domain name space. Currently, the host sri-nic.arpa is used. (This command is a synonym for the **lserver sri-nic.arpa**.) The name of the root server can be changed with the **set root** command.

finger [*name*] [> *filename*]
finger [*name*] [>> *filename*]

Connects with the finger server on the current host. The current host is defined when a previous lookup for a host was successful and returned address information. (See the **set querytype=A** command.)

Name is optional. > and >> can be used to redirect output in the usual manner.

ls *domain* [> *filename*]
ls *domain* [>> *filename*]
ls –a *domain* [> *filename*]
ls –a *domain* [>> *filename*]
ls –h *domain* [> *filename*]
ls –h *domain* [>> *filename*]
ls –d *domain* [> *filename*]

List the information available for *domain*. The default output contains host names and their Internet addresses. The **–a** option lists aliases of hosts in the domain. The **–h** option lists CPU and operating system information for the domain. The **–d** option lists all contents of a zone transfer. When output is directed to a file, hash marks are printed for every 50 records received from the server.

view *filename*

Sorts and lists the output of previous **ls** command(s) with **more**(1).

help

? Prints a brief summary of commands.

set *keyword* [=*value*]

This command is used to change state information that affects the lookups. Valid keywords are:

all Prints the current values of the various options to set. Information about the current default server and host is also printed.

[no]debug

Turn debugging mode on. A lot more information is printed about the packet sent to the server and the resulting answer. (Default = nodebug, abbreviation = [no]deb)

[no]d2 Turn exhaustive debugging mode on. Essentially all fields of every packet are printed.
(Default = nod2)

[no]defname

Append the default domain name to every lookup.
(Default = defname, abbreviation = [no]def)

[no]search

With defname, search for each name in parent domains of the current domain.
(Default = search)

domain=*name*

Change the default domain name to *name*. The default domain name is appended to all lookup requests if the **defname** option

has been set. The search list is set to parents of the domain with at least two components in their names.
(Default = value in hostname or */etc/resolv.conf*, abbreviation = do)

querytype=*value*

type=*value*
Change the type of information returned from a query to one of:

A	the host's Internet address (the default).
CNAME	the canonical name for an alias.
HINFO	the host CPU and operating system type.
MD	the mail destination.
MX	the mail exchanger.
MG	the mail group member.
MINFO	the mailbox or mail list information.
MR	the mail rename domain name.
NS	nameserver for the named zone.

Other types specified in the RFC883 document are valid but are not very useful.
(Abbreviation = q)

[no]recurse
Tell the name server to query other servers if it does not have the information.
(Default = recurse, abbreviation = [no]rec)

retry=*number*
Set the number of retries to *number*. When a reply to a request is not received within a certain amount of time (changed with **set timeout**), the request is resent. The retry value controls how many times a request is resent before giving up.
(Default = 2, abbreviation = ret)

root=*host*
Change the name of the root server to *host*. This affects the **root** command.
(Default = sri-nic.arpa, abbreviation = ro)

timeout=*number*
Change the time-out interval for waiting for a reply to *number* seconds.
(Default = 10 seconds, abbreviation = t)

[no]vc Always use a virtual circuit when sending requests to the server.
(Default = novc, abbreviation = [no]v)

DIAGNOSTICS

If the lookup request was not successful, an error message is printed. Possible errors are:

Time-out
: The server did not respond to a request after a certain amount of time (changed with **set timeout**=*value*) and a certain number of retries (changed with **set retry**=*value*).

No information
: Depending on the query type set with the **set querytype** command, no information about the host was available though the host name is valid.

Nonexistent domain
: The host or domain name does not exist.

Connection refused
Network is unreachable
: The connection to the name or finger server could not be made at the current time. This error commonly occurs with finger requests.

Server failure
: The name server found an internal inconsistency in its database and could not return a valid answer.

Refused
: The name server refused to service the request.

The following error should not occur, and it indicates a bug in the program.

Format error
: The name server found that the request packet was not in the proper format.

FILES

/etc/resolv.conf initial domain name and name server addresses.

SEE ALSO

resolver(3N), resolver(5), named(1M), RFC882, RFC883

NAME

ping – send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

ping [**–v**] *host* [*count*] [*packetsize*]

DESCRIPTION

ping utilizes the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams ("pings") have an IP and ICMP header, followed by a **struct timeval**, and then an arbitrary number of *pad* bytes used to fill out the packet. The default datagram length is 64 bytes, but this may be changed using the command-line option described below:

–v Verbose output. ICMP packets other than ECHO RESPONSE that are received are listed.

When using **ping** for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be ''pinged.'' **ping** sends one datagram and prints one line of output. If an optional *count* is given, that number of requests is sent, and and one line for every ECHO_RESPONSE returned is printed. No output is produced if there is no response. Round-trip times and packet loss statistics are computed. When all responses have been received or the program times out (*count* packets sent), or if the program is terminated with a SIGINT, a brief summary is displayed.

This program is intended for use in network testing, measurement and management. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is unwise to use **ping** during normal operations or from automated scripts.

NAME

rcp – remote file copy

SYNOPSIS

rcp [**–p**] file1 file2
rcp [**–p**] [**–r**] file ... directory

DESCRIPTION

rcp copies files between machines. Each *file* or *directory* argument is either a remote file name of the form “rhost:path”, or a local file name (containing no “:” characters, or a “/” before any “:”s).

If the **–r** option is specified and any of the source files are directories, **rcp** copies each subtree rooted at that name; in this case the destination must be a directory. Note that using the **–r** option and specifying a particular existing directory on the destination machine as the recipient of the transfer causes a directory with the source directory name to be created under the existing destination directory. For example, executing the command **rcp -r** 1:X 2:A/ to transfer the contents of directory X on host 1 to existing directory A on host 2 causes a directory X to be created under directory A on host 2. In other words, the files under directory X on host 1 will be placed under directory /A/X on host 2. Executing the **rcp -r** 1:X 2:A/ command when directory A does *not* exist on host 2 causes a new directory A to be created on host 2. The contents of directory X on host 1 are then placed under the newly created directory A.

By default, the mode and owner of *file2* are preserved if it already existed; otherwise, the mode of the source file modified by the **umask**(2) on the destination host is used. The **–p** option causes **rcp** to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the **umask.**

If *path* is not a full path name, it is interpreted relative to your login directory on *rhost*. A *path* on a remote host may be quoted (using \, ", or ') so that the metacharacters are interpreted remotely.

rcp does not prompt for passwords; your current local user name must exist on *rhost* and allow remote command execution via **resh**(1C).

rcp handles third-party copies, where neither source nor target files are on the current machine. Hostnames may also take the form “rname@rhost” to use *rname* rather than the current user name on the remote host.

SEE ALSO

cp(1), ftp(1C), resh(1C), rlogin(1C)

BUGS

Doesn’t detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

Is confused by any output generated by commands in a *.login*, *.profile*, or *.cshrc* file on the remote host.

The destination user and hostname may have to be specified as “rhost.rname” when the destination machine is running the 4.2 BSD version of **rcp**.

Executing **rcp** file1 *host* :file2 where *host* is the machine on which the command is being executed can cause unpredictable results.

NAME

rdump – file system dump across the network

SYNOPSIS

/etc/rdump [key [*argument* ...] filesystem]

DESCRIPTION

The **rdump** utility copies to magnetic tape all files changed after a certain date in the *filesystem*. The command is identical in operation to **dump**(8), except the *f* key should be specified and the file supplied should be of the form *machine:device*.

The **rdump** utility creates a remote server, **/etc/rmt**, on the client machine to access the tape device.

SEE ALSO

dump(8), rmt(1M)

DIAGNOSTICS

Same as **dump**(8) with a few extra related to the network.

BUGS

High density tapes (i.e. 6250 BPI) are by default written with 32*1024 byte records. Older versions (including 4.2bsd) of **/etc/rmt**, only allow 10*1024 read or write operations, thus causing **rdump** to fail. Use the "b 10" option to **rdump**(1C) to get around this problem.

NAME

resh – remote shell

SYNOPSIS

resh host [**–l** username] [**–n**] command
host [**–l** username] [**–n**] command

DESCRIPTION

resh connects to the specified *host*, and executes the specified *command*. **resh** copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; **resh** normally terminates when the remote command does.

The remote username used is the same as your local username, unless you specify a different remote name with the **–l** option. This remote name must be equivalent (in the sense of **rlogin**(1C)) to the originating account; no provision is made for specifying a password with a command.

If you omit *command*, then instead of executing a single command, you will be logged in on the remote host using **rlogin**(1C).

Shell metacharacters which are not quoted are interpreted on local machine, while quoted metacharacters are interpreted on the remote machine. Thus, the command

```
resh otherhost cat remotefile >> localfile
```

appends the remote file *remotefile* to the localfile *localfile*, while

```
resh otherhost cat remotefile ">>" otherremotefile
```

appends *remotefile* to *otherremotefile*.

Host names are given in the file */etc/hosts* or retrieved through the name server. Each host has one standard name (the first name given in the file), which is rather long and unambiguous and, optionally, one or more nicknames. The host names for local machines are also commands in the directory */usr/hosts*; if you put this directory in your search path, **resh** can be omitted.

FILES

/etc/hosts
*/usr/hosts/**

SEE ALSO

rlogin(1C)

BUGS

If you are using **csh**(1) and put a **resh**(1C) in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. If no input is desired, you should redirect the input of **resh** to */dev/null* using the **–n** option.

You cannot run an interactive command (like **vi**(1)); use **rlogin**(1C).

Stop signals stop the local **resh** process only.

NAME

rlogin – remote login

SYNOPSIS

rlogin rhost [**–e***c*] [**–8**] [**–L**] [**–l** username]
rhost [**–e***c*] [**–8**] [**–L**] [**–l** username]

DESCRIPTION

rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost.*

Each host has a file */etc/hosts.equiv* which contains a list of *rhost*'s with which it shares account names. (The host names must be the standard names as described in **resh**(1C).) When you **rlogin** as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file *.rhosts* in his login directory. Each line in this file should contain an *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user is not equivalent to the remote user, a login and password will be prompted for on the remote machine as in **login**(1). To avoid some security problems, the *.rhosts* file must be owned by either the remote user or root.

The remote terminal type is the same as your local terminal type (as given in your environment TERM variable). The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. The local terminal speed is also copied to the remote system if the server supports the option. All echoing takes place at the remote site, so that (except for delays) the rlogin is transparent. Flow control via ^S and ^Q and flushing of input and output on interrupts are handled properly. The optional argument **–8** allows an eight-bit input data path at all times; otherwise, parity bits are stripped except when the remote side's stop and start characters are other than ^S/^Q.

The argument **–L** allows the rlogin session to be run in litout mode. A line of the form "~." disconnects from the remote host, where "~" is the escape character. Similarly, the line "~^Z" (where ^Z, control-Z, is the suspend character) will suspend the rlogin session. Substitution of the delayed-suspend character (normally ^Y) for the suspend character suspends the send portion of the rlogin, but allows output from the remote system. A different escape character may be specified by the **–e** option. There is no space separating this option flag and the argument character.

The local hosts terminal type and speed are propagated during the **rlogin** session.

SEE ALSO

resh(1C) login(1)

FILES

*/usr/hosts/**	for *rhost* version of the command
/etc/hosts.equiv	for a list of *rhosts* shared account names

NAME

route – manually manipulate the routing tables

SYNOPSIS

/etc/route [**–f**] [**–n**] [*command args*]

DESCRIPTION

Route is a program used to manually manipulate the network routing tables.

Route accepts two commands: *add* to add a route, and *delete* to delete a route.

All commands have the following syntax:

/etc/route *command* [**net** | **host**] *destination gateway* [*metric*]

where *destination* is a host or network, *gateway* is the next-hop gateway to which packets should be addressed, and *metric* is a count indicating the number of hops to the *destination*. The metric is required for *add* commands; it must be zero if the destination is on a directly-attached network, and nonzero if the route utilizes one or more gateways. If adding a route with metric 0, the gateway given is the address of this host on the common network, indicating the interface to be used for transmission. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. The optional keywords **net** and **host** force the destination to be interpreted as a network or a host, respectively. Otherwise, if the *destination* has a "local address part" of INADDR_ANY, or if the *destination* is the symbolic name of a network, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. If the route is to a destination connected via a gateway, the *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first as a host name using *gethostbyname*(3N). If this lookup fails, *getnetbyname*(3N) is then used to interpret the name as that of a network. All addresses must be specified in dotted-decimal form.

Route uses the */ev/ip* multiplexor with the IP_ADD_RT and IP_DEL_RT *ioctl*'s to do its work. As such, only the super-user may modify the routing tables.

If the **–f** option is specified, *route* will "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

The **–n** option prevents attempts to print host and network names symbolically when reporting actions.

DIAGNOSTICS

add [host | network] [host | network name]: gateway [gateway name] flags [route flags]

The specified route is being added to the tables. The values printed are from the routing table entry supplied in the *ioctl* call. If the gateway address used was not the primary address of the gateway (the first one returned by *gethostbyname*), the gateway address is printed numerically as well as symbolically.

delete [host | network] [host | network name]: gateway [gateway name] flags [route flags]

As above, but when deleting an entry.

Network is unreachable
An attempt to add a route failed because the gateway listed was not on a directly-connected network. The next-hop gateway must be given.

not in table
A delete operation was attempted for an entry which wasn't present in the tables.

NAME

rrestore – restore a file system dump across the network

SYNOPSIS

/etc/rrestore [key [name ...]]

DESCRIPTION

The **rrestore** command obtains from magnetic tape files saved by a previous **dump**(8). The command is identical in operation to **restore**(8) except the *f* key should be specified and the file supplied should be of the form *machine:device*.

The **rrestore** utility creates a remote server, **/etc/rmt**, on the client machine to access the tape device.

SEE ALSO

restore(8), rmt(1M)

DIAGNOSTICS

Same as **restore**(8) with a few extra related to the network.

NAME

ruptime – show host status of local machines

SYNOPSIS

ruptime [**-ar**] [**-lut**]

DESCRIPTION

ruptime gives a status line like **uptime** for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 11 minutes are shown as being down.

Users idle an hour or more are not counted unless the **-a** flag is given.

Normally, the listing is sorted by host name. The **-l** , **-t** , and **-u** flags specify sorting by load average, uptime, and number of users, respectively. The **-r** flag reverses the sort order.

FILES

*/usr/spool/rwho/whod.**	data files

SEE ALSO

rwho(1C)

BUGS

Packet size limitations in the *rwhod* protocol prevent the distribution of more than 41 users per system.

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [**–a**]

DESCRIPTION

The **rwho** command produces output similar to **who,** but for all machines on the local network. If no report has been received from a machine for 5 minutes, **rwho** assumes the machine is down and does not report users last known to be logged into that machine.

If a users hasn't typed to the system for a minute or more, **rwho** reports this idle time. If a user hasn't typed to the system for an hour or more, the user will be omitted from the output of **rwho** unless the **–a** flag is specified.

FILES

*/usr/spool/rwho/whod.**	information about other machines

SEE ALSO

ruptime(1C), rwhod(1M)

BUGS

This is unwieldy when the number of machines on the local net is large.

Packet size limitations in the **rwhod** protocol prevent the distribution of more than 41 users per system.

NAME

talk – talk to another user

SYNOPSIS

talk person [ttyname]

DESCRIPTION

talk is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on your own machine, *person* is simply the person's login name. If you wish to talk to a user on another host, *person* is of the form *user@host*.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name, where *ttyname* is of the form "ttyXX."

When first called, **talk** sends the message:

```
Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine
```

to the user you want to talk to. At this point, the recipient of the message should reply by typing

```
talk  your_name@your_machine
```

It doesn't matter from which machine the recipient replies, as long as his login name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing control L (^L) causes the screen to be reprinted, while your erase, kill, and word-kill characters behave normally. To exit, type your interrupt character; **talk** then moves the cursor to the bottom of the screen and restores the terminal to its previous state.

Permission to talk may be denied or granted by use of the mesg command. At the outset, talking is allowed. Certain commands, in particular **nroff** and **pr**, disallow messages in order to prevent messy output.

FILES

/etc/hosts	to find the recipient's machine
/etc/utmp	to find the recipient's tty

SEE ALSO

mail(1), mesg(1), who(1), write(1)

BUGS

The version of **talk**(1) released with DYNIX/ptx uses a protocol that is based on the 4.3 BSD protocol, which is incompatible with the 4.2 BSD protocol.

NAME

telnet – user interface to the TELNET protocol

SYNOPSIS

telnet [host [port]]

DESCRIPTION

telnet is used to communicate with another host using the **TELNET** protocol. If **telnet** is invoked without arguments, it enters command mode, indicated by its prompt ("telnet>"). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command (see below) with those arguments.

Once a connection has been opened, **telnet** enters an input mode. The input mode entered will be either "character at a time" or "line by line," depending on what the remote system supports.

In "character at a time" mode, most text typed is immediately sent to the remote host for processing.

In "line by line" mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The "local echo character" (initially "^E") may be used to turn off and on the local echo. (This would mostly be used to enter passwords without the password being echoed.)

In either mode, if the *localchars* toggle is TRUE (the default in line mode--see below), the user's **quit**, **intr**, and **flush** characters are trapped locally and sent as TELNET protocol sequences to the remote side. The **toggle** *autoflush* and **toggle** *autosynch* options (see below) cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of **quit** and **intr**).

While connected to a remote host, **telnet** command mode may be entered by typing the **telnet** "escape character" (initially "^]"). When in command mode, normal terminal editing conventions are available.

COMMANDS

The following commands are available. Only enough of each command to uniquely identify it need be typed. (This is also true for arguments to the **mode**, **set**, **toggle**, and **display** commands.)

open *host* [*port*]

Open a connection to the named host. If no port number is specified, **telnet** will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see **hosts**(5)) or an Internet address specified in the "dot notation" (see **inet**(3N)).

close

Close a TELNET session, and return to command mode.

quit

Close any open TELNET session, and exit **telnet**. An end-of-file (in command mode) also closes a session and exits.

z

Suspend **telnet**. This command only works when the user is using the **csh**(1).

mode *type*

type is either *line* (for "line by line" mode) or *character* (for "character at a time" mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

status

Show the current status of **telnet**. This includes the peer one is connected to, as well as the current mode.

display [*argument...*]

Displays all or some of the **set** and **toggle** values (see below).

? [*command*]

Get help. With no arguments, **telnet** prints a help summary. If a command is specified, **telnet** prints the help information for just that command.

send *arguments*

Sends one or more special character sequences to the remote host. The following arguments may be specified (more than one argument may be specified at a time):

escape

Sends the current **telnet** escape character (initially "^]").

synch

Sends the TELNET SYNCH sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system. If it doesn't work, a lower case "r" may be echoed on the terminal).

brk

Sends the TELNET BRK (Break) sequence, which may have significance to the remote system.

ip

Sends the TELNET IP (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

ao

Sends the TELNET AO (Abort Output) sequence, which should cause the remote system to flush all output **from** the remote system **to** the user's terminal.

ayt

Sends the TELNET AYT (Are You There) sequence, to which the remote system may or may not choose to respond.

ec

Sends the TELNET EC (Erase Character) sequence, which should cause the remote system to erase the last character entered.

el

Sends the TELNET EL (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

ga

Sends the TELNET GA (Go Ahead) sequence, which likely has no significance to the remote system.

nop

Sends the TELNET NOP (No OPeration) sequence.

?

Prints out help information for the **send** command.

set *argument value*

Set any one of a number of **telnet** variables to a specific value. The special value "off" turns off the function associated with the variable. The values of variables may be interrogated with the **display** command. The variables which may be specified are:

echo

This is the value (initially "^E") which, when in "line by line" mode, toggles between doing local echoing of entered characters (for normal processing) and suppressing echoing of entered characters (for entering a password, for example).

escape

This is the **telnet** escape character (initially "^[") which causes entry into **telnet** command mode (when connected to a remote system).

interrupt

If **telnet** is in *localchars* mode (see **toggle** *localchars* below) and the *interrupt* character is typed, a TELNET IP sequence (see **send** *ip* above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's **intr** character.

quit

If **telnet** is in *localchars* mode (see **toggle** *localchars* below) and the *quit* character is typed, a TELNET BRK sequence (see **send** *brk* above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's *quit* character.

flushoutput

If **telnet** is in *localchars* mode (see **toggle** *localchars* below) and the *flushoutput* character is typed, a TELNET AO sequence (see **send** *ao* above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's *flush* character.

erase

If **telnet** is in *localchars* mode (see **toggle** *localchars* below), and if **telnet** is operating in "character at a time" mode, then when this character is typed, a TELNET EC sequence (see **send** *ec* above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's *erase* character.

kill

If **telnet** is in *localchars* mode (see **toggle** *localchars* below), and if **telnet** is operating in "character at a time" mode, then when this character is typed, a TELNET EL sequence (see **send** *el* above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's *kill* character.

eof

If **telnet** is operating in "line by line" mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's *eof* character.

toggle *arguments...*

Toggle (between TRUE and FALSE) various flags that control how **telnet** responds to events. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are:

localchars

If this is TRUE, then the *flush*, *interrupt*, *quit*, *erase*, and *kill* characters (see **set** above) are recognized locally, and transformed into (hopefully) appropriate TELNET control sequences (respectively *ao*, *ip*, *brk*, *ec*, and *el*; see **send** above). The initial value for this toggle is TRUE in "line by line" mode, and FALSE in "character at a time" mode.

autoflush

If *autoflush* and *localchars* are both TRUE, then when the *ao*, *intr*, or *quit* characters are recognized (and transformed into TELNET sequences; see **set** above for details), **telnet** refuses to display any data on the user's terminal until the remote system acknowledges (via a TELNET *Timing Mark* option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user had not done an "stty noflsh"; otherwise it is FALSE (see **stty(1)**).

autosynch

If *autosynch* and *localchars* are both TRUE, then when either the *intr* or *quit* character is typed (see **set** above for descriptions of the *intr* and *quit* characters), the resulting TELNET sequence sent is followed by the TELNET SYNCH sequence. This procedure should cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

crmod

Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage returns but never line feeds. The initial value for this toggle is FALSE.

debug

Toggles endpoint level debugging (useful only to the superuser). The initial value for this toggle is FALSE.

options

Toggles the display of some internal **telnet** protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.

netdata

Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

?

Displays the legal **toggle** commands.

BUGS

There is no adequate way for dealing with flow control.

On some remote systems, echo has to be turned off manually when in "line by line" mode.

There is enough settable state to justify a *.telnetrc* file.

No capability for a *.telnetrc* file is provided.

In "line by line" mode, the terminal's *eof* character is only recognized (and sent to the remote system) when it is the first character on a line.

NAME

tftp – trivial file transfer program

SYNOPSIS

tftp [host]

DESCRIPTION

tftp is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case **tftp** uses *host* as the default host for future transfers (see the **connect** command below).

COMMANDS

Once **tftp** is running, it issues the prompt "tftp>" and recognizes the following commands:

connect *host-name* [*port*]

Set the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers; thus, the **connect** command does not actually create a connection but merely remembers what host is to be used for transfers. You do not have to use the **connect** command; the remote host can be specified as part of the **get** or **put** commands.

mode *transfer-mode*

Set the mode for transfers; *transfer-mode* may be *ascii* or *binary*. The default is *ascii*.

put *file*

put *localfile remotefile*
put *file1 file2 ... fileN remote-directory*

Put a file or set of files to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be a UNIX machine.

get *filename*
get *remotename localname*
get *file1 file2 ... fileN*

Get a file or set of files from the specified *sources*. *Source* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter form is used, the last hostname specified becomes the default for future transfers.

quit Exit **tftp**. An end of file also exits.

verbose

Toggle verbose mode.

trace Toggle packet tracing.

status Show current status.

rexmt *retransmission-timeout*
Set the per-packet retransmission time-out, in seconds.

timeout *total-transmission-timeout*
Set the total transmission timeout, in seconds.

ascii Shorthand for "mode ascii."

binary
Shorthand for "mode binary."

? [*command-name* ...]
Print help information.

BUGS

Because there is no user-login or validation within the TFTP protocol, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site and therefore difficult to document here.

NAME

ftpd – DARPA Internet File Transfer Protocol server

SYNOPSIS

/etc/ftpd [**–d**] [**–l**] [**–t**timeout]

DESCRIPTION

ftpd is the DARPA Internet File Transfer Protocol server process. The server uses the TCP protocol and listens at the port specified in the "ftp" service specification; see **services**(5).

If the **–d** option is specified, debugging information is written to the syslog.

If the **–l** option is specified, each ftp session is logged in the syslog.

The ftp server will time out an inactive session after 15 minutes. If the **–t** option is specified, the inactivity timeout period will be set to *timeout*.

The ftp server currently supports the following ftp requests; case is not distinguished.

Request	**Description**
ABOR	abort previous command
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CDUP	change to parent of current working directory
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory ("ls -lg")
MKD	make a directory
MODE	specify data transfer *mode*
NLST	give name list of files in directory ("ls")
NOOP	do nothing
PASS	specify password
PASV	prepare for server-to-server transfer
PORT	specify data connection port
PWD	print the current working directory
QUIT	terminate session
RETR	retrieve a file
RMD	remove a directory
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
STOU	store a file with a unique name
STRU	specify data transfer *structure*
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

The remaining ftp requests specified in Internet RFC 959 are recognized but not implemented.

The ftp server will abort an active file transfer only when the ABOR command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in Internet RFC 959.

ftpd interprets file names according to the “globbing” conventions used by **csh**(1). This allows users to utilize the metacharacters “*?[]{}~”.

ftpd authenticates users according to three rules.

1) The user name must be in the password data base, */etc/passwd*, and not have a null password. In this case, a password must be provided by the client before any file operations may be performed.

2) The user name must not appear in the file */etc/ftpusers*.

3) The user must have a standard shell returned by **getusershell**(3).

4) If the user name is “anonymous” or “ftp”, an anonymous ftp account must be present in the password file (user “ftp”). In this case, the user is allowed to log in by specifying any password. (By convention, this is given as the client host’s name).

In the last case, **ftpd** takes special measures to restrict the client’s access privileges. The server performs a **chroot**(2) command to the home directory of the “ftp” user. In order that system security is not breached, it is recommended that the “ftp” subtree be constructed with care. The following rules are recommended:

~ftp) Make the home directory owned by “ftp” and unwritable by anyone.

~ftp/bin)
Make this directory owned by the superuser and unwritable by anyone. The program **ls**(1) must be present to support the list commands. This program should have mode 111.

~ftp/etc)
Make this directory owned by the superuser and unwritable by anyone. The files **passwd**(5) and **group**(5) must be present for the **ls** command to work properly. These files should be mode 444.

~ftp/pub)
Make this directory mode 777 and owned by “ftp”. Users should then place files which are to be accessible via the anonymous account in this directory.

SEE ALSO

ftp(1C), syslogd(1M)

BUGS

The anonymous account is inherently dangerous and should be avoided when possible.

The server must run as the superuser to create tli endpoints with privileged port numbers. It maintains an effective user id of the logged-in user, reverting to the superuser only when binding addresses to tli endpoints. The possible security holes have been extensively scrutinized but are possibly incomplete.

NAME

inetd – internet "super–server"

SYNOPSIS

/etc/inetd [**–d**] [configuration file]

DESCRIPTION

inetd should be run at boot time by */etc/rc2.d/S50netservers* . It then listens for connections on certain internet tli endpoints. When a connection is found on one of its endpoints, it decides to what service the port corresponds and invokes a program to service the request. After the program is finished, it continues to listen on the endpoint (except in some cases which are described below). Essentially, **inetd** allows running one daemon to invoke several others, reducing load on the system.

Upon execution, **inetd** reads its configuration information from a configuration file which, by default, is */etc/inetd.conf*. There must be an entry for each field of the configuration file, with entries for each field separated by a tab or a space. Comments are denoted by a "#" at the beginning of a line. There must be an entry for each field. The fields of the configuration file are as follows:

service name
tli endpoint type
protocol
wait/nowait
user
server program
server program arguments

The *service name* entry is the name of a valid service in the file */etc/services/* . For "internal" services (discussed below), the service name **must** be the official name of the service (that is, the first entry in */etc/services*).

The *tli endpoint type* should be one of "stream", "dgram", "raw", "rdm", or "seqpacket", depending on whether the tli endpoint is a stream, datagram, raw, reliably delivered message, or sequenced packet endpoint.

The *protocol* must be a valid protocol as given in */etc/protocols* . Examples might be "tcp" or "udp".

The *wait/nowait* entry is applicable to datagram endpoints only. (Other endpoints should have a "nowait" entry in this space.) If a datagram server connects to its peer, freeing the socket so **inetd** can received further messages on the socket, it is said to be a "multi-threaded" server and should use the "nowait" entry. For datagram servers which process all incoming datagrams on a socket and eventually time out, the server is said to be "single-threaded" and should use a "wait" entry. "Comsat" ("biff") and "talk" are both examples of the latter type of datagram server. **tftpd** is an exception; it is a datagram server that establishes pseudoconnections. It must be listed as "wait" in order to avoid a race; the server reads the first packet, creates a new socket, and then forks and exits to allow **inetd** to check for new service requests to spawn new servers.

The *user* entry should contain the name of the user whom the server should run as. This allows for servers to be given less permission than root. The *server program* entry should contain the pathname of the program which is to be executed by **inetd** when a request is found on its socket. If **inetd** provides this service internally, this entry should be "internal".

The arguments to the server program should be just as they normally are, starting with argv[0], which is the name of the program. If the service is provided internally, the word "internal" should take the place of this entry.

inetd provides several "trivial" services internally by use of routines within itself. These services are "echo", "discard", "chargen" (character generator), "daytime" (human readable time), and "time" (machine readable time, in the form of the number of seconds since midnight, January 1, 1900). All of these services are tcp based. For details of these services, consult the appropriate RFC from the Network Information Center.

inetd rereads its configuration file when it receives a hangup signal, SIGHUP. Services may be added, deleted, or modified when the configuration file is reread.

SEE ALSO

comsat(1M), ftpd(1M), rexecd(1M), rlogind(1M), reshd(1M), telnetd(1M), tftpd(1M)

NAME

named – Internet domain name server

SYNOPSIS

/etc/named [**–d** *debuglevel*] [**–p** *port#*] [{**–b**} *bootfile*]

DESCRIPTION

named is the Internet domain name server. (See RFC883 for more information on the Internet name-domain system.) Without any arguments, **named** will read the default boot file */etc/named.boot*, read any initial data, and listen for queries.

Options are:

–d Print debugging information. A number after the "d" determines the level of messages printed.

–p Use a different port number. The default is the standard port number as listed in */etc/services*.

–b Use an alternate boot file. This is optional and allows you to specify a file with a leading dash.

Any additional argument is taken as the name of the boot file. The boot file contains information about where the name server is to get its initial data. If multiple boot files are specified, only the last is used. Lines in the boot file cannot be continued on subsequent lines. The following is a small example:

```
;
;         boot file for name server
;
directory        /usr/local/domain

; type      domain                 source host/file             backup file

cache       .                                                   root.cache
primary     Berkeley.EDU           berkeley.edu.zone
primary     32.128.IN-ADDR.ARPA    ucbhosts.rev
secondary   CC.Berkeley.EDU        128.32.137.8 128.32.137.3 cc.zone.bak
secondary   6.32.128.IN-ADDR.ARPA 128.32.137.8 128.32.137.3 cc.rev.bak
primary     0.0.127.IN-ADDR.ARPA                                localhost.rev
forwarders 10.0.0.78 10.2.0.78
; slave
```

The "directory" line causes the server to change its working directory to the directory specified. This can be important for the correct processing of $INCLUDE files in primary zone files.

The "cache" line specifies that data in "root.cache" is to be placed in the backup cache. Its main use is to specify data such as locations of root domain servers. This cache is not used during normal operation but is used as "hints" to find the current root servers. The file "root.cache" is in the same format as "berkeley.edu.zone". There can be more than one "cache" file specified. The cache files are processed in such a way as to preserve the time-to-live's of data dumped out. Data for the root nameservers is kept artificially valid if necessary.

The first "primary" line states that the file *berkeley.edu.zone* contains authoritative data for the "Berkeley.EDU" zone. The file *berkeley.edu.zone* contains data in the master file format described in RFC883. All domain names are relative to the origin, in this case, "Berkeley.EDU". (See below for a more detailed description).

The second "primary" line states that the file *ucbhosts.rev* contains authoritative data for the domain "32.128.IN-ADDR.ARPA," which is used to translate addresses in network 128.32 to hostnames. Each master file should begin with an SOA record for the zone (see below).

The first "secondary" line specifies that all authoritative data under "CC.Berkeley.EDU" is to be transferred from the name server at 128.32.137.8. If the transfer fails, it will try 128.32.137.3 and continue trying the addresses, up to 10, listed on this line. The secondary copy is also authoritative for the specified domain. The first nondotted-quad address on this line will be taken as a filename in which to back up the transferred zone. The name server will load the zone from this backup file if it exists when it boots, providing a complete copy even if the master servers are unreachable. Whenever a new copy of the domain is received by automatic zone transfer from one of the master servers, this file will be updated.

The second "secondary" line states that the address-to-hostname mapping for the subnet 128.32.136 should be obtained from the same list of master servers as the previous zone.

The "forwarders" line specifies the addresses of sitewide servers that will accept recursive queries from other servers. If the boot file specifies one or more forwarders, the server will send all queries for data not in the cache to the forwarders first. Each forwarder will be asked in turn until an answer is returned or the list is exhausted. If no answer is forthcoming from a forwarder, the server will continue as it would have without the forwarders line unless it is in "slave" mode. The forwarding facility is useful to cause a large sitewide cache to be generated on a master and to reduce traffic over links to outside servers. It can also be used to allow servers to run that do not have access directly to the Internet, but wish to act as though they do.

The "slave" line (shown commented out) is used to put the server in slave mode. In this mode, the server will only make queries to forwarders. This option is normally used on machines that wish to run a server but, for physical or administrative reasons, cannot be given access to the Internet. They do, however, have access to a host that does have access to the Internet.

The "sortlist" line can be used to indicate networks that are to be preferred over other, unlisted networks. Queries for host addresses from hosts on the same network as the server will receive responses with local network addresses listed first, then addresses on the sort list, then other addresses. This line is only acted on at initial startup. When reloading the nameserver with a SIGHUP, this line is ignored.

The master file consists of control information and a list of resource records for objects in the zone of the forms:

```
$INCLUDE <filename> <opt_domain>
$ORIGIN <domain>
```

```
<domain> <opt_ttl> <opt_class> <type> <resource_record_data>
```

where *domain* is "." for root, "@" for the current origin, or a standard domain name. If *domain* is a standard domain name that does not end with “.”, the current origin is appended to the domain. Domain names ending with “.” are unmodified.

The *opt_domain* field is used to define an origin for the data in an included file. It is equivalent to placing a $ORIGIN statement before the first line of the included file. The field is optional. Neither the *opt_domain* field nor $ORIGIN statements in the included file modify the current origin for this file.

The *opt_ttl* field is an optional integer number for the time-to-live field. It defaults to zero, meaning the minimum value specified in the SOA record for the zone.

The *opt_class* field is the object address type; currently, only one type is supported, IN, for objects connected to the DARPA Internet.

The *type* field contains one of the following tokens; the data expected in the *resource_record_data* field is in parentheses.

A	a host address (dotted quad)
NS	an authoritative name server (domain)
MX	a mail exchanger (domain)
CNAME	the canonical name for an alias (domain)
SOA	marks the start of a zone of authority (domain of originating host, domain address of maintainer, a serial number and the following parameters in seconds: refresh, retry, expire and minimum TTL (see RFC883))
MB	a mailbox domain name (domain)
MG	a mail group member (domain)
MR	a mail rename domain name (domain)
NULL	a null resource record (no format or data)
WKS	a well known service description (not implemented yet)
PTR	a domain name pointer (domain)
HINFO	host information (cpu_type OS_type)
MINFO	mailbox or mail list information (request_domain error_domain)

Resource records normally end at the end of a line but may be continued across lines between opening and closing parentheses. Comments are introduced by semicolons and continue to the end of the line.

Each master zone file should begin with an SOA record for the zone. An example SOA record is as follows:

```
@       IN      SOA     ucbvax.Berkeley.EDU. rwh.ucbvax.Berkeley.EDU. (
                                2.89    ; serial
                                10800   ; refresh
                                3600    ; retry
                                3600000         ; expire
86400 ) ; minimum
```

The SOA lists a serial number, which should be changed each time the master file is changed. Secondary servers check the serial number at intervals specified by the refresh time in seconds. If the serial number changes, a zone transfer will be done to load the new data. If a master server cannot be contacted when a refresh is due, the retry time specifies the interval at which refreshes should be attempted until successful. If a master server cannot be contacted within the interval given by the expire time, all data from the zone is discarded by secondary servers. The minimum value is the time-to-live used by records in the file with no explicit time-to-live value.

NOTES

The boot file directives "domain" and "suffixes" have been obsoleted by a more useful resolver-based implementation of suffixing for partially qualified domain names. The prior mechanisms could fail under a number of situations, especially when the local nameserver did not have complete information.

The following signals have the specified effect when sent to the server process using the **kill**(1) command.

SIGHUP
Causes server to read *named.boot* and reload database.

SIGINT
Dumps current data base and cache to */usr/tmp/named_dump.db*

SIGIOT
Dumps statistics data into */usr/tmp/named.stats* if the server is compiled -DSTATS. Statistics data is appended to the file.

SIGSYS
Dumps the profiling data in */usr/tmp* if the server is compiled with profiling (server forks, chdirs and exits).

SIGTERM
Dumps the primary and secondary database files. Used to save modified data on shutdown if the server is compiled with dynamic updating enabled.

SIGUSR1
Turns on debugging; each SIGUSR1 increments debug level. (SIGEMT on older systems without SIGUSR1)

SIGUSR2
Turns off debugging completely. (SIGFPE on older systems without SIGUSR2)

FILES

/etc/named.boot	name server configuration boot file
/etc/named.pid	the process id
/usr/tmp/named.run	debug output
/usr/tmp/named_dump.db	dump of the name server database
/usr/tmp/named.stats	nameserver statistics data

NAME

netd – network daemon

SYNOPSIS

/etc/netd [**–nt**] [config_file | –]

DESCRIPTION

The **netd** daemon configures a *STREAMS* network from a specification given in a configuration file whose format is described in **netconf**(4).

By default the configuration is taken from the file **/etc/netconf**. An alternate file may be specified as a command line parameter (*config_file*). The special filename "–" is taken to mean the standard input.

The **–t** option may be used to debug a configuration file. It produces trace information indicating the sequence of opens, links, etc. performed by **netd** to create the *STREAMS* architecture defined by the configuration.

The **–n** option, which also implies **–t** , inhibits the actual building of the *STREAMS* network.

A typical configuration file for ptx/TCP/IP might contain the following network description. (See **netconf**(4) for a description of the syntax.)

```
tcp     dc     /dev/tcp
udp     dc     /dev/udp
ip      dc     /dev/ip
arp     m       arp
eth     dc     /dev/eth
echo    dc     /dev/eth
loop    d      /dev/loop

%%

tcp      ip
udp      ip
ip       arp      IP_NET={89.0.0.3,,forwb=n,kalive=y,576}
arp      eth      ARP_TYPE=notrailers
ip       loop     IP_NET={127.0.0.1,,,}
echo     eth      ECHO_TYPE
```

The following control actions (as described in **netconf**(4)) are currently understood:

IP_NET={*internet-address, subnet-mask,* **forwb**=*boolean,* **kalive**=*boolean, mtu*}

This specifies the Internet address and subnet mask for a stream below IP. It also specifies whether broadcasts are to be forwarded from other networks to that network, whether keepalive should be on for that network, and a value to override the network's **Maximum Transmission Unit** (MTU).

This is typically used when a network is multiplexed below an **IP** (Internet Protocol) driver. The Internet address and Subnet mask are given in standard Internet dotted format, and may be omitted (supplied as a null string) when not relevant, in which case it will be passed as zero. The *broadcast* flag is specified as a boolean value, i.e. y or n (for "yes"/"true" — "do forward" — or "no"/"false" — "don't forward"), or may be omitted in the defaulting case (don't forward). The *keepalive* flag is specified as a boolean value, i.e. y or n (for "yes"/"true" — "do use keepalive" — or "no"/"false" — "don't use keepalive"), or may be omitted in the defaulting case (don't use keepalive). The *mtu* value is specified as an unsigned integer. If the value is 0 or omitted, then the MTU indicated by the lower driver will be used. The MTU should be less that that specified by the lower driver.

(**Note:** A broadcast packet for transmission to a network either originates from the host or has previously been received from another network. The first type is always transmitted. The second type is only transmitted if the forward broadcast flag ("forwb") is set to "y". This flag is only relevant for a gateway.)

In cases where only the network address is supplied, the syntax

IP_NET=*internet-address*

may be used.

ARP_TYPE=*trailers*

This specifies that this stream is to be registered with the network driver below as the stream on which incoming ARP packets are to be received.

In addition, *trailers* may take the value trailers or notrailers (the default), to specify whether trailer encapsulated packets should be sent on the network interface.

ECHO_TYPE

This specifies that this stream is to be registered with the network driver below as the stream on which incoming XEROX ECHO packets are to be received.

SEE ALSO

netconf(4), hosts(4).

NAME

reshd – remote shell server

SYNOPSIS

reshd [–aln]

DESCRIPTION

reshd is the server for the **rcmd**(3) routine and, consequently, for the **resh**(1) program. The server provides remote execution facilities with authentication based on privileged port numbers from trusted hosts.

reshd is invoked by the internet server (see **inetd**(8)), for service requests at the port indicated in the "cmd" service specification; see **services**(5). When a service request is received, the following protocol is initiated:

1) The server checks the client's source port. If the port is not in the range 512 through 1023, the server aborts the connection.
2) The server reads characters from the tli endpoint up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.
3) If the number received in step 2 is nonzero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 512 through 1023.
4) The server checks the client's source address and requests the corresponding host name (see **gethostbyaddr**(3), **hosts**(5), and *named* (8)). If the hostname cannot be determined, the dot-notation representation of the host address is used. If the hostname is in the same domain as the server (according to the last two components of the domain name), or if the **–a** option is given, the addresses for the hostname are requested, verifying that the name and address correspond. If address verification fails, the connection is aborted with the message, "Host address mismatch."
5) A null-terminated user name of at most 16 characters is retrieved on the initial tli endpoint. This user name is interpreted as the user identity on the client's machine.
6) A null-terminated user name of at most 16 characters is retrieved on the initial tli endpoint. This user name is interpreted as a user identity to use on the server 's machine.
7) A null-terminated command to be passed to a shell is retrieved on the initial tli endpoint. The length of the command is limited by the upper bound on the size of the system's argument list.
8) **reshd** then validates the user using **ruserok**(3), which uses the file */etc/hosts.equiv* and the *.rhosts* file found in the user's home directory. The **–l** option prevents **ruserok**(3) from doing any validation based on the user's *.rhosts* file, unless the user is the superuser.
9) A null byte is returned on the initial tli endpoint, and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by **reshd**.

Transport-level keepalive messages are enabled unless the **–n** option is present. The use of keepalive messages allows sessions to be timed out if the client crashes or becomes unreachable.

DIAGNOSTICS

Except for the last one listed below, all diagnostic messages are returned on the initial tli endpoint, after which any network connections are closed. An error is indicated by a leading byte with a value of 1. (Zero is returned in step 9, above, upon successful completion of all the steps prior to the execution of the login shell.)

"locuser too long"
The name of the user on the client's machine is longer than 16 characters.

"remuser too long"
The name of the user on the remote machine is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"No remote directory."
The **chdir** command to the home directory failed.

"Permission denied."
The authentication procedure described above failed.

"Can't make pipe."
The pipe needed for the **stderr**, wasn't created.

"Can't fork; try again."
A **fork** by the server failed.

"<shellname>: ..."
The user's login shell could not be started. This message is returned on the connection associated with the **stderr**, and is not preceded by a flag byte.

SEE ALSO

resh(1C), rcmd(3X), ruserok(3X)

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

A more extensible protocol (such as Telnet) should be used.

NAME

rexecd – remote execution server

SYNOPSIS

/etc/rexecd

DESCRIPTION

rexecd is the server for the **rexec (3X)** routine. The server provides remote execution facilities with authentication based on user names and encrypted passwords.

rexecd is invoked by the internet server (see **inetd**(8)), for service requests at the port indicated in the "exec" service specification; see **services (5).** When a service request is received the following protocol is initiated:

1) The server reads characters from the endpoint up to a null ("\0") byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is nonzero, it is interpreted as the port number of a secondary stream to be used for the **stderr.** A second connection is then created to the specified port on the client's machine.

3) A null terminated user name of at most 16 characters is retrieved on the initial tli endpoint.

4) A null terminated password of at most 16 characters is retrieved on the initial tli endpoint.

5) A null terminated command to be passed to a shell is retrieved on the initial tli endpoint. The length of the command is limited by the upper bound on the size of the system's argument list.

6) **rexecd** then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.

7) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by **rexecd .**

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr,** after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

"username too long"
The name is longer than 16 characters.

"password too long"
The password is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"Password incorrect."
The wrong was password supplied.

"No remote directory."
The **chdir** command to the home directory failed.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

BUGS

Indicating "Login incorrect" as opposed to "Password incorrect" is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rlmod – rlogin protocol module

DESCRIPTION

The **rlmod** module is responsible for presenting a streamttty interface to **ld0**. It presents a transport packet interface downstream to tcp.

rlmod handles the TIOCSWINSZ and TIOCGWINSZ ioctls that are passed downstream by **ld0**. If the window size is changed, an expedited data message is sent to the remote system. This message is a TIOCPKT_WINDOW rlogin protocol message. Additionally, a SIGWINCH signal is sent upstream to notify programs that the window size has been changed. If the window size is changed by the client, a rlogin control message is received by the rlogin protocol module and a SIGWINCH signal is generated if the received size information differs from the current size information.

If a M_FLUSH FLUSHW or FLUSHRW message is sent downstream to the rlogin module, it sends a TIOCPKT_FLUSHWRITE expedited message to the remote system.

SEE ALSO

rlogin(1C)

BUGS

Currently, the TIOCPKT_NOSTOP message is sent when **ld0** sends either a MC_STOP or MC_START message downstream. **rlmod** should be changed to send a TIOCPKT_DOSTOP and handle disabling and reenabling the queues for proper operation of the MC_STOP and MC_START control messages.

NAME

rlogind – remote login server

SYNOPSIS

/etc/rlogind [–aln]

DESCRIPTION

rlogind is the server for the **rlogin**(1C) program. The server provides a remote login facility with authentication based on privileged port numbers from trusted hosts.

rlogind is invoked by the internet server (see **inetd**(8)), normally for requests to connect to the rlogin port as indicated by the /etc/services file (see **services**(5)).

The server checks the client's source address and requests the corresponding host name (see **gethostbyaddr**(3), **hosts**(5), and **named**(1M)). If the hostname cannot be determined, the dot-notation representation of the host address is used.

Once the source port and address have been checked, **rlogind** pushes the telnet protocol module (see **rlmod**(1M)) and the line discipline module (see **ld**(7)) onto the transport stream. The major and minor numbers of the transport stream deteremine a "terminal name" (see dev_to_ttyname) that is used to create the utmp entry. The utmp entry is used to represent the "tty" associated with the virtual terminal.

rlogind then validates the user according to the following steps:

1) The local (server-end) user name is looked up in the password file. If the lookup fails, the login name is prompted with a standard login. Super-user logins are not allowed.

2) The server checks the client's source address and requests the corresponding hostname (see **gethostbyaddr**, **hosts**(5),and **named**(1M)). If the hostname cannot be determined, the dot notation representation of the host address is used. If the hostname is in the same domain as the server (according to the last two components of the domain name), or if the -a option is given, the addresses for the hostname are requested, verifying that the name and address correspond. If address verification fails, the password will be requested. If address verification fails, the message "Host address mismatch" will be printed.

3) If the user is not the super-user (user id 0), the file **/etc/hosts.equiv** is consulted for a list of hosts considered "equivalent."

4) If the client's host name is present in this file, the authentication is considered successful.

5) If the lookup fails, then the file .rhosts in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. The –l option prevents ruserok from doing any validation based on the user's .rhosts file.

6) If this lookup fails, the connection is terminated.

The **rlogind** does not participate in the normal flow of data, but waits for the death of the shell to clean up the umtp entry.

SEE ALSO

rlogin(1C)

NAME

rmt – remote magtape protocol module

SYNOPSIS

/etc/rmt

DESCRIPTION

Rmt is a program used by the remote dump and restore programs in manipulating a magnetic tape drive through an interprocess communication connection. The **rmt** program is normally started up with an **rexec**(3X) or **rcmd**(3X) call.

The **rmt** program accepts requests specific to the manipulation of magnetic tapes, performs the commands, then responds with a status indication. All responses are in ASCII and in one of two forms. Successful commands have responses of

A*number*\n

where *number* is an ASCII representation of a decimal number. Unsuccessful commands are responded to with

E*error-number*\n*error-message*\n,

where *error-number* is one of the possible error numbers described in **intro**(2) and *error-message* is the corresponding error string as printed from a call to **perror**(3). The protocol is comprised of the following commands (a space is present between each token).

O device mode
Open the specified *device* using the indicated *mode*. *Device* is a full pathname and *mode* is an ASCII representation of a decimal number suitable for passing to **open**(2). If a device had already been opened, it is closed before a new open is performed.

C device
Close the currently open device. The *device* specified is ignored.

L whence offset
Perform an **lseek**(2) operation using the specified parameters. The response value is that returned from the **lseek** call.

W count
Write data onto the open device. The **rmt** server reads *count* bytes from the connection, aborting if a premature end-of-file is encountered. The response value is that returned from the **write**(2) call.

R count
Read *count* bytes of data from the open device. The **rmt** command performs the requested **read**(2) and responds with A*count-read*\n if the read was successful; otherwise an error in the standard format is returned. If the read was successful, the data read is then sent.

I operation count
Perform a MTIOCOP **ioctl**(2) command using the specified parameters. The parameters are interpreted as the ASCII representations of the decimal values to place in the *mt_op* and *mt_count* fields of the structure used in the **ioctl** call.

The return value is the *count* parameter when the operation is successful.

S	Return the status of the open device, as obtained with a MTIOCGET **ioctl** call. If the operation was successful, an "ack" is sent with the size of the status buffer, then the status buffer is sent (in binary).

Any other command causes **rmt** to exit.

DIAGNOSTICS

All responses are of the form described above.

SEE ALSO

rcmd(3X), rexec(3X), mtio(4), rdump(1C), rrestore(1C)

NAME

rwhod – system status server

SYNOPSIS

/etc/rwhod

DESCRIPTION

rwhod is the server which maintains the database used by the **rwho**(1C) and **ruptime**(1C) programs. Its operation is predicated on the ability to broadcast messages on a network.

rwhod operates as both a producer and consumer of status information. As a producer of information, it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other **rwhod** servers' status messages, validating them, and then recording them in a collection of files located in the directory */usr/spool/rwho*.

The server transmits and receives messages at the port indicated in the "rwho" service specification; see **services**(5). The messages sent and received are of the form:

```
struct  outmp {
        char    out_line[8];/* tty name */
        char    out_name[8];/* user id */
        long    out_time;/* time on */
};

struct  whod {
        char    wd_vers;
        char    wd_type;
        char    wd_fill[2];
        int     wd_sendtime;
        int     wd_recvtime;
        char    wd_hostname[32];
        int     wd_loadav[3];
        int     wd_boottime;
        struct  whoent {
                struct  outmp we_utmp;
                int     we_idle;
        } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the **w**(1) program and represent load averages over the 5-, 10-, and 15-minute intervals prior to a server's transmission; they are multiplied by 100 for representation in an integer. The host name included is that returned by the **gethostname**(2) system call, with any trailing domain name omitted. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the **utmp**(5) entry for each nonidle terminal line and a value indicating the time in seconds since a character was last received on the terminal line.

Messages received by the **rwho** server are discarded unless they originated at an **rwho** server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by **rwhod** are placed in files named *whod.hostname* in the directory */usr/spool/rwho*. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 3 minutes. **rwhod** performs an **nlist**(3) on */unix* every 30 minutes to guard against the possibility that this file is not the system image currently operating.

SEE ALSO

rwho(1C), ruptime(1C)

BUGS

There should be a way to relay status information between networks. Status information should be sent only upon request rather than continuously. People often interpret the server dying or network communication failures as a machine going down.

NAME

sendmail – send mail over the internet

SYNOPSIS

/usr/lib/sendmail [flags] [address ...]

/usr/lib/newaliases

/usr/lib/mailq [–v]

DESCRIPTION

sendmail sends a message to one or more *recipients*, routing the message over whatever networks are necessary. **sendmail** does internetwork forwarding as necessary to deliver the message to the correct place.

sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; **sendmail** is used only to deliver preformatted messages.

With no flags, **sendmail** reads its standard input up to an end-of-file or a line consisting only of a single dot and sends a copy of the message found there to all of the addresses listed. It determines the network(s) to use based on the syntax and contents of the addresses.

Local addresses are looked up in a file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash. Normally, the sender is not included in any alias expansions; e.g., if "john" sends to "group", and "group" includes "john" in the expansion, the letter will not be delivered to "john".

Flags are:

–ba	Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the "From:" and "Sender:" fields are examined for the name of the sender.
–bd	Run as a daemon. This requires Berkeley IPC. **sendmail** will fork and run in background listening on tli endpoint 25 for incoming SMTP connections. This is normally run from */etc/initd/netservers*.
–bi	Initialize the alias database.
–bm	Deliver mail in the usual way (default).
–bp	Print a listing of the queue.
–bs	Use the SMTP protocol as described in RFC821 on standard input and output. This flag implies all the operations of the **–ba** flag that are compatible with SMTP.
–bt	Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.
–bv	Verify names only – do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.

–bz	Create the configuration freeze file.
–C*file*	Use alternate configuration file. **sendmail** refuses to run as root if an alternate configuration file is specified. The frozen configuration file is bypassed.
–d*X*	Set debugging value to *X*.
–F*fullname*	Set the full name of the sender.
–f*name*	Sets the name of the "from" person (i.e., the sender of the mail). **–f** can only be used by "trusted" users (normally *root, daemon,* and *network)* or if the person you are trying to become is the same as the person you are.
–h*N*	Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop. If not specified, "Received:" lines in the message are counted.
–n	Don't do aliasing.
–o*x value*	Set option *x* to the specified *value*. Options are described below.
–q[*time*]	Processed saved messages in the queue at given intervals. If *time* is omitted, process the queue once. *time* is given as a tagged number, with "s" being seconds, "m" being minutes, "h" being hours, "d" being days, and "w" being weeks. For example, "–q1h30m" or "–q90m" would both set the timeout to one hour thirty minutes. If *time* is specified, **sendmail** will run in background. This option can be used safely with **–bd**.
–k*name*	An alternate and obsolete form of the **–f** flag.
–t	Read message for recipients. To:, Cc:, and Bcc: lines will be scanned for recipient addresses. The Bcc: line will be deleted before transmission. Any addresses in the argument list will be suppressed, that is, they will not receive copies even if listed in the message header.
–v	Go into verbose mode. Alias expansions will be announced, etc.

There are also a number of processing options that may be set. Normally, these will only be used by a system administrator. Options may be set either on the command line using the **–o** flag or in the configuration file. These are described in detail in the *Sendmail Installation and Operation Guide*. The options are:

A*file*	Use alternate alias file.
c	On mailers that are considered "expensive" to connect to, don't initiate immediate connection. This requires queueing.
d*x*	Set the delivery mode to *x*. Delivery modes are "i" for interactive (synchronous) delivery, "b" for background

	(asynchronous) delivery, and "q" for queue only – i.e., actual delivery is done the next time the queue is run.
D	Try to automatically rebuild the alias database if necessary.
e*x*	Set error processing to mode *x*. Valid modes are "m" to mail back the error message; "w" to "write" back the error message (or mail it back if the sender is not logged in); "p" to print the errors on the terminal (default); "q" to throw away error messages (only exit status is returned); and "e" to do special processing for the BerkNet. If the text of the message is not mailed back by modes "m" or "w" and if the sender is local to this machine, a copy of the message is appended to the file "dead.letter" in the sender's home directory.
F*mode*	The mode to use when creating temporary files.
f	Save UNIX-style "From" lines at the front of messages.
g*N*	The default group id to use when calling mailers.
H*file*	The SMTP help file.
i	Do not take dots on a line by themselves as a message terminator.
L*n*	The log level.
m	Send to "me" (the sender) also if I am in an alias expansion.
o	If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (i.e., commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.
Q*queuedir*	Select the directory in which to queue messages.
r*timeout*	The timeout on reads; if none is set, **sendmail** will wait forever for a mailer. This option violates the word (if not the intent) of the SMTP specification, so the timeout should probably be fairly large.
S*file*	Save statistics in the named file.
s	Always instantiate the queue file, even under circumstances where it is not strictly necessary. This provides safety against system crashes during delivery.
T*time*	Set the timeout on undelivered messages in the queue to the specified time. After delivery has failed (e.g., because of a host being down) for this amount of time, failed messages will be returned to the sender. The default is three days.
t*stz,dtz*	Set the name of the time zone.

u*N*	Set the default user id for mailers.

In aliases, the first character of a name may be a vertical bar to cause interpretation of the rest of the name as a command to pipe the mail to. It may be necessary to quote the name to keep **sendmail** from suppressing the blanks from between arguments. For example, a common alias is:

```
msgs: " | /usr/lib/msgs -s"
```

Aliases may also have the syntax ":include:*filename*" to ask **sendmail** to read the named file for a list of recipients. For example, an alias such as:

```
poets: ":include:/usr/local/lib/poets.list"
```

would read */usr/local/lib/poets.list* for the list of addresses making up the group.

sendmail returns an exit status describing what it did. The codes are defined in *<sysexits.h>*

EX_OK	Successful completion on all addresses.
EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as "cannot fork."
EX_NOHOST	Host name not recognized.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

If invoked as **newaliases, sendmail** will rebuild the alias database. If invoked as **mailq, sendmail** will print the contents of the mail queue.

FILES

Except for */usr/lib/sendmail.cf*, these pathnames are all specified in */usr/lib/sendmail.cf*. Thus, these values are only approximations.

/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.pag	
/usr/lib/aliases.dir	data base of alias names
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	help file
/usr/lib/sendmail.st	collected statistics
/usr/spool/mqueue/*	temp files

SEE ALSO

mail(1), rmail(1), syslog(3), rc(4);
DARPA Internet Request For Comments RFC819, RFC821, RFC822;
Sendmail – An Internetwork Mail Router (SMM:16);
Sendmail Installation and Operation Guide (SMM:7)

NAME

syslogd – log systems messages

SYNOPSIS

/etc/syslogd [**–f***configfile*] [**–m***markinterval*] [**–d**]

DESCRIPTION

syslogd reads and logs messages into a set of files described by the configuration file */etc/syslog.conf*. Each message is one line. A message can contain a priority code, marked by a number in angle braces at the beginning of the line. Priorities are defined in <*sys/syslog.h*>. **syslogd** reads from the streams pipe */dev/syslog*.

syslogd configures when it starts up and whenever it receives a hangup signal. Lines in the configuration file have a *selector* to determine the message priorities to which the line applies and an *action*. The *action* field is separated from the selector by one or more tabs.

Selectors are semicolon-separated lists of priority specifiers. Each priority has a *facility* describing the part of the system that generated the message, a dot, and a *level* indicating the severity of the message. Symbolic names may be used. An asterisk selects all facilities. All messages of the specified level or higher (greater severity) are selected. More than one facility may be selected, using commas to separate them. For example:

```
*.emerg;mail,daemon.crit
```

Selects all facilities at the *emerg* level and the *mail* and *daemon* facilities at the *crit* level.

Known facilities and levels recognized by **syslogd** are those listed in **syslog**(3) without the leading "LOG_." The additional facility "mark" has a message at priority LOG_INFO sent to it every 20 minutes. (This may be changed with the **–m** flag.) The "mark" facility is not enabled by a facility field containing an asterisk. The level "none" may be used to disable a particular facility. For example,

```
*.debug;mail.none
```

Sends all messages except mail messages to the selected file.

The second part of each line describes where the message is to be logged if this line is selected. There are four forms:

- A filename (beginning with a leading slash). The file will be opened in append mode.
- A hostname preceded by an at sign ("@"). Selected messages are forwarded to the **syslogd** on the named host.
- A comma separated list of users. Selected messages are written to those users if they are logged in.
- An asterisk. Selected messages are written to all logged-in users.

Blank lines and lines beginning with "#" are ignored.

For example, the configuration file:

```
kern,mark.debug         /dev/console
*.notice;mail.info      /usr/spool/adm/syslog
```

```
*.crit                    /usr/adm/critical
kern.err                  @ucbarpa
*.emerg                   *
*.alert                   eric,kridle
*.alert;auth.warning      ralph
```

logs all kernel messages and 20-minute marks onto the system console, all notice (or higher) level messages, and all mail system messages except debug messages into the file */usr/spool/adm/syslog*, and all critical messages into */usr/adm/critical*. Kernel messages of error severity or higher are forwarded to ucbarpa. All users will be informed of any emergency messages, the users "eric" and "kridle" will be informed of any alert messages, and the user "ralph" will be informed of any alert message, or any warning message (or higher) from the authorization system.

The flags are:

–f	Specify an alternate configuration file.
–m	Select the number of minutes between mark messages.
–d	Turn on debugging.

syslogd creates the file */etc/syslog.pid*, if possible, containing a single line with its process id. This can be used to kill or reconfigure **syslogd.**

To bring **syslogd** down, it should be sent a terminate signal (e.g., kill `cat */etc/syslog.pid*`).

FILES

/etc/syslog.conf	the configuration file
/etc/syslog.pid	the process id
/dev/log	Name of the UNIX domain datagram log tli endpoint

SEE ALSO

syslog(3)

NAME

talkd – remote user communication server

SYNOPSIS

/etc/talkd

DESCRIPTION

talkd is the server that notifies a user that somebody else wants to initiate a conversation. It acts as a repository of invitations, responding to requests by clients wishing to rendezvous to hold a conversation. In normal operation, a client, the caller, initiates a rendezvous by sending a CTL_MSG to the server of type LOOK_UP (see <*protocols/talkd.h*>). This causes the server to search its invitation tables to check if an invitation currently exists for the caller (to speak to the callee specified in the message).

If the lookup fails, the caller sends an ANNOUNCE message causing the server to broadcast an announcement on the callee's login ports requesting contact. When the callee responds, the local server uses the recorded invitation to respond with the appropriate rendezvous address, and the caller and callee client programs establish a stream connection through which the conversation takes place.

SEE ALSO

talk(1C), write(1)

NAME

telnetd – DARPA TELNET protocol server

SYNOPSIS

/etc/telnetd

DESCRIPTION

telnetd is a server that provides support for the DARPA standard TELNET virtual terminal protocol. **telnetd** is invoked by the internet server (see **inetd**(1M)), normally for requests to connect to the TELNET port as indicated by the */etc/services* file (see **services**(5)).

telnetd operates by pushing the telnet protocol module (see **tnmod**(1M)) and the line discipline module (see **ld**) onto the transport stream. The major and minor number of the transport stream determine a "terminal name" (see dev_to_ttyname) which is used to create the utmp entry which is used to represent the "tty" associated with the virtual terminal.

telnetd requests the tnmod protocol engine to indicate willingness to do remote echo of characters and to suppress go ahead. The virtual terminal is configured to operate in canonical mode.

SEE ALSO

telnet(1C)

BUGS

Some TELNET commands are only partially implemented.

Binary mode has no common interpretation except between similar operation systems (Unix, in this case).

telnetd never sends TELNET go ahead commands.

NAME

tftpd – DARPA Trivial File Transfer Protocol server

SYNOPSIS

/etc/tftpd

DESCRIPTION

tftpd is a server which supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the "tftp" service description; see **services**(5). The server is normally started by **inetd**(8).

The use of **tftp** does not require an account or password on the remote system. Due to the lack of authentication information, **tftpd** will allow only publicly readable files to be accessed. Files may be written only if they already exist and are publicly writable. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network. This may not be appropriate on all systems, and its implications should be considered before enabling tftp service. The server should have the user ID with the lowest possible privilege.

SEE ALSO

tftp(1C), inetd(1M)

NAME

tnmod – telnet protocol module

DESCRIPTION

The **tnmod** module is responsible for presenting a **streamtty** interface to **ld0**. It presents a transport packet interface downstream to tcp.

tnmod implements the TELNET protocol. Initial option negotiation is started via the TN_OPTS ioctl message.

This module is pushed onto the transport stream by **telnetd**.

SEE ALSO

telnet(1C)

BUGS

tnmod does not currently support the terminal type suboption.

NAME

syslog, openlog, closelog, setlogmask – control system log

SYNOPSIS

#include <syslog.h>

openlog(ident, logopt, facility)
char *ident;

syslog(priority, message, parameters ...)
char *message;

closelog()

setlogmask(maskpri)

DESCRIPTION

syslog arranges to write *message* onto the system log maintained by **syslogd**(8). The message is tagged with *priority*. The message looks like a **printf**(3) string, except that **%m** is replaced by the current error message (collected from *errno*). A trailing newline is added if needed. This message will be read by **syslogd**(8) and written to the system console, log files, or forwarded to **syslogd** on another host as appropriate.

Priorities are encoded as a *facility* and a *level*. The facility describes the part of the system generating the message. The level is selected from an ordered list:

LOG_EMERG	A panic condition. This is normally broadcast to all users.
LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database.
LOG_CRIT	Critical conditions (e.g., hard device errors).
LOG_ERR	Errors.
LOG_WARNING	Warning messages.
LOG_NOTICE	Conditions that are not error conditions but should possibly be handled specially.
LOG_INFO	Informational messages.
LOG_DEBUG	Messages that contain information normally of use only when debugging a program.

If **syslog** cannot pass the message to **syslogd**, it will attempt to write the message on */dev/console* if the LOG_CONS option is set (see below).

If special processing is needed, **openlog** can be called to initialize the log file. *ident* is a string that is prepended to every message. *logopt* is a bit-field indicating logging options. Current values for *logopt* are:

LOG_PID	Log the process id with each message: useful for identifying instantiations of daemons.
LOG_CONS	Force writing messages to the console if unable to send it to **syslogd.** This option is safe to use in daemon processes that have no controlling terminal since **syslog** will fork before opening the console.

LOG_NDELAY	Open the connection to **syslogd** immediately. Normally the open is delayed until the first message is logged. Useful for programs that need to manage the order in which file descriptors are allocated.
LOG_NOWAIT	Don't wait for children forked to log messages on the console. This option should be used by processes that enable notification of child termination via SIGCHLD, as **syslog** may otherwise block waiting for a child whose exit status has already been collected.

facility encodes a default facility to be assigned to all messages that do not have an explicit facility encoded:

LOG_KERN	Messages generated by the kernel. These cannot be generated by any user processes.
LOG_USER	Messages generated by random user processes. This is the default facility identifier if none is specified.
LOG_MAIL	The mail system.
LOG_DAEMON	System daemons, such as **ftpd**(8), **routed**(8), etc.
LOG_AUTH	The authorization system: **login**(1), **su**(1), **getty**(8), etc.
LOG_LPR	The line printer spooling system: **lpr**(1), **lpc**(8), **lpd**(8), etc.
LOG_LOCAL0	Reserved for local use. Similarly for LOG_LOCAL1 through LOG_LOCAL7.

closelog can be used to close the log file.

setlogmask sets the log priority mask to *maskpri* and returns the previous mask. Calls to **syslog** with a priority not set in *maskpri* are rejected. The mask for an individual priority *pri* is calculated by the macro LOG_MASK(*pri*); the mask for all priorities up to and including *toppri* is given by the macro LOG_UPTO(*toppri*). The default allows all priorities to be logged.

EXAMPLES

```
syslog(LOG_ALERT, “who: internal error 23”);

openlog(“ftpd”, LOG_PID, LOG_DAEMON);
setlogmask(LOG_UPTO(LOG_ERR));
syslog(LOG_INFO, “Connection from host %d”, CallingHost);

syslog(LOG_INFO | LOG_LOCAL2, “foobar error: %m”);
```

SEE ALSO

syslogd(1M)

NAME

htonl, htons, ntohl, ntohs – convert values between host and network byte order

SYNOPSIS

#include <sys/types.h>
#include <netinet/in.h>

netlong = htonl(hostlong);
ulong netlong, hostlong;

netshort = htons(hostshort);
ushort netshort, hostshort;

hostlong = ntohl(netlong);
ulong hostlong, netlong;

hostshort = ntohs(netshort);
ushort hostshort, netshort;

DESCRIPTION

These routines convert 16 and 32 bit quantities between network byte order and host byte order. Since National 32Ks, Intel *86s, DEC Vaxs, and Motorola 68Ks use a different byte order, these routines are needed to communicate between them.

These routines are most often used in conjunction with Internet addresses and ports as returned by **getservent**(3N).

SEE ALSO

getservent(3N)

NAME

gethostbyaddr, gethostbyname, sethostent, endhostent – get network host entry

SYNOPSIS

```
#include <netdb.h>

struct hostent *gethostbyname(name)
char *name;

struct hostent *gethostbyaddr(addr, len, type)
char *addr; int len, type;

sethostent(stayopen)
int stayopen

endhostent()
```

DESCRIPTION

gethostbyname and **gethostbyaddr** each return a pointer to an object with the following structure containing the broken-out fields of a line in the network host data base, */etc/hosts*:

```
struct  hostent {
        char    *h_name;        /* official name of host */
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* address type */
        int     h_length;       /* length of address */
        char    *h_addr;        /* address */
};
```

The members of this structure are:

h_name	Official name of the host.
h_aliases	A zero-terminated array of alternate names for the host.
h_addrtype	The type of address being returned; currently always AF_INET.
h_length	The length, in bytes, of the address.
h_addr	A pointer to the network address for the host. Host addresses are returned in network byte order.

sethostent opens and rewinds the file. If the *stayopen* flag is nonzero, the host data base will not be closed after each call to **gethostent** (either directly, or indirectly through one of the other gethost calls).

endhostent closes the file.

gethostbyname and **gethostbyaddr** sequentially search from the beginning of the file until a matching host name or host address is found, or until EOF is encountered. Host addresses are supplied in network order.

FILES

/etc/hosts
/etc/*domainname*/hosts.byname
/etc/*domainname*/hosts.byaddr

SEE ALSO

hosts(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.

NAME

getmyinaddr – returns current IP and port address for local endpoint

SYNOPSIS

```
getmyinaddr (fd, sin, sinlen)
int fd;
struct sockaddr_in *sin;
int sinlen;
```

DESCRIPTION

getmyinadrr returns the current IP and port address for the local endpoint. **fd** refers to an open UDP or TCP endpoint. **sin** refers to a buffer large enough to hold a *sockaddr_in* format address. **sinlen** should be initialized to **sizeof** (struct *sockaddr_in*).

getmyinaddr returns 0 on success –1 on failure.

NAME

getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent – get network entry

SYNOPSIS

#include <netdb.h>

struct netent *getnetent()

struct netent *getnetbyname(name)
char *name;

struct netent *getnetbyaddr(net)
int net;

setnetent(stayopen)
int stayopen

endnetent()

DESCRIPTION

getnetent, **getnetbyname**, and **getnetbyaddr** each return a pointer to an object with the following structure containing the broken-out fields of a line in the network database, */etc/networks*:

```
struct  netent {
        char    *n_name;        /* official name of net */
        char    **n_aliases;    /* alias list */
        int     n_addrtype;     /* net number type */
        long    n_net;          /* net number */
};
```

The members of this structure are:

n_name	The official name of the network.
n_aliases	A zero-terminated list of alternate names for the network.
n_addrtype	The type of the network number returned; currently only AF_INET.
n_net	The network number. Network numbers are returned in machine byte order.

getnetent reads the next line of the file, opening the file if necessary.

setnetent opens and rewinds the file. If the *stayopen* flag is nonzero, the net data base will not be closed after each call to **getnetent** (either directly, or indirectly through one of the other getnet calls).

endnetent closes the file.

getnetbyname and **getnetbyaddr** sequentially search from the beginning of the file until a matching net name or net address is found, or until EOF is encountered. Network numbers are supplied in host order.

FILES

/etc/networks

SEE ALSO

networks(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only Internet network numbers are currently understood. Expecting network numbers to fit in no more than 32 bits is probably naive.

NAME

getpeerinaddr – returns current IP and port address for remote endpoint

SYNOPSIS

```
getpeerinaddr (fd, sin, sinlen)
int fd;
struct sockaddr_in *sin;
int sinlen;
```

DESCRIPTION

getpeerinaddr returns the current IP and port address for the remote endpoint. **fd** refers to an open UDP or TCP endpoint. **sin** refers to a buffer large enough to hold a *sockaddr_in* format address. **sinlen** should be initialized to **sizeof** (struct *sockaddr_in*).

getpeerinaddr returns 0 on success –1 on failure.

NAME

getservent, getservbyport, getservbyname, setservent, endservent – get service entry

SYNOPSIS

```
#include <netdb.h>

struct servent *getservent()

struct servent *getservbyname(name, proto)
char *name, *proto;

struct servent *getservbyport(port, proto)
int port; char *proto;

setservent(stayopen)
int stayopen

endservent()
```

DESCRIPTION

getservent, **getservbyname**, and **getservbyport** each return a pointer to an object with the following structure containing the broken-out fields of a line in the network services data base, */etc/services*:

```
struct  servent {
        char    *s_name;        /* official name of service */
        char    **s_aliases;    /* alias list */
        long    s_port;         /* port service resides at */
        char    *s_proto;       /* protocol to use */
};
```

The members of this structure are:

s_name	The official name of the service.
s_aliases	A zero-terminated list of alternate names for the service.
s_port	The port number at which the service resides. Port numbers are returned in network byte order.
s_proto	The name of the protocol to use when contacting the service.

getservent reads the next line of the file, opening the file if necessary.

setservent opens and rewinds the file. If the *stayopen* flag is nonzero, the net data base will not be closed after each call to **getservent** (either directly, or indirectly through one of the other getserv calls).

endservent closes the file.

getservbyname and **getservbyport** sequentially search from the beginning of the file until a matching protocol name or port number is found or until EOF is encountered. If a protocol name is also supplied (non-NULL), searches must also match the protocol.

FILES

/etc/services

SEE ALSO

services(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Expecting port numbers to fit in a 32 bit quantity is probably naive.

NAME

inet_addr, inet_network, inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof – Internet address manipulation routines

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

u_long inet_addr(cp)
char *cp;

u_long inet_network(cp)
char *cp;

char *inet_ntoa(in)
struct in_addr in;

struct in_addr inet_makeaddr(net, lna)
int net, lna;

int inet_lnaof(in)
struct in_addr in;

int inet_netof(in)
struct in_addr in;
```

DESCRIPTION

The routines *inet_addr* and *inet_network* each interpret character strings representing numbers expressed in the Internet standard dot (.) notation, returning numbers suitable for use as Internet addresses and Internet network numbers, respectively. The routine *inet_ntoa* takes an Internet address and returns an ASCII string representing the address in dot (.) notation. The routine *inet_makeaddr* takes an Internet network number and a local network address and constructs an Internet address from it. The routines *inet_netof* and *inet_lnaof* break apart Internet host addresses, returning the network number and local network address part, respectively.

All Internet address are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

INTERNET ADDRESSES

Values specified using the dot (.) notation take one of the following forms:

a.b.c.d
a.b.c
a.b
a

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on a Sequent system the bytes referred to above appear as d.c.b.a. That is, Sequent machine bytes are ordered from right to left.

When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three part address format convenient for specifying Class B network addresses as 128.net.host.

When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as net.host.

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied as parts in a . notation may be decimal, octal, or hexadecimal, as specified in the C language A leading 0x or 0X implies hexadecimal and a leading 0 implies octal. Otherwise, the number is interpreted as decimal.

SEE ALSO

gethostent(3N), getnetent(3N), hosts(5), networks(5),

DIAGNOSTICS

The value –1 is returned by *inet_addr* and *inet_network* for malformed requests.

BUGS

The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed. The string returned by *inet_ntoa* resides in a static memory area.

NAME

res_mkquery, res_send, res_init, dn_comp, dn_expand – resolver routines

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

res_mkquery(op, dname, class, type, data, datalen, newrr, buf,
buflen)
int op;
char *dname;
int class, type;
char *data;
int datalen;
struct rrec *newrr;
char *buf;
int buflen;

res_send(msg, msglen, answer, anslen)
char *msg;
int msglen;
char *answer;
int anslen;

res_init()

dn_comp(exp_dn, comp_dn, length, dnptrs, lastdnptr)
char *exp_dn, *comp_dn;
int length;
char **dnptrs, **lastdnptr;

dn_expand(msg, eomorig, comp_dn, exp_dn, length)
char *msg, *eomorig, *comp_dn, exp_dn;
int length;
```

DESCRIPTION

These routines are used for making, sending, and interpreting packets for use with Internet domain name servers. Global information that is used by the **resolver** routines is kept in the variable *_res*. Most of the values have reasonable defaults and can be ignored. The following options are stored in *_res.options* and are defined in *resolv.h*. (Options are stored in a simple bit mask containing the bitwise OR of the options enabled.)

RES_INIT

True if the initial name server address and default domain name are initialized (i.e., **res_init** has been called).

RES_DEBUG

Print debugging messages.

RES_AAONLY

Accept authoritative answers only. With this option, **res_send** should continue until it finds an authoritative answer or finds an error. Currently this is not implemented.

RES_USEVC
Use TCP connections for queries instead of UDP datagrams.

RES_STAYOPEN
Used with RES_USEVC to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used.

RES_IGNTC
Unused currently (ignore truncation errors, i.e., don't retry with TCP).

RES_RECURSE
Set the recursion-desired bit in queries. This is the default. (**res_send** does not do iterative queries and expects the name server to handle recursion.)

RES_DEFNAMES
If set, **res_mkquery** will append the default domain name to single-component names (those that do not contain a dot). This is the default.

RES_DNSRCH
If this option is set, the standard host lookup routine **gethostbyname** will search for host names in the current domain and in parent domains.

res_init reads the initialization file to get the default domain name and the Internet address of the initial hosts running the name server. If this line does not exist, the host running the resolver is tried. **res_mkquery** makes a standard query message and places it in *buf*. **res_mkquery** will return the size of the query or –1 if the query is larger than *buflen*. *op* is usually QUERY but can be any of the query types defined in *nameser.h*. *dname* is the domain name. If *dname* consists of a single label and the RES_DEFNAMES flag is enabled (the default), the current domain name will be appended to *dname*. The current domain name is defined by the hostname or is specified in a system file; it can be overridden by the environment variable LOCALDOMAIN. *newrr* is currently unused but is intended for making update messages.

res_send sends a query to name servers and returns an answer. It will call **res_init** if RES_INIT is not set, send the query to the local name server, and handle timeouts and retries. The length of the message is returned or –1 if there were errors.

dn_expand expands the compressed domain name *comp_dn* to a full domain name. Expanded names are converted to upper case. *msg* is a pointer to the beginning of the message, *exp_dn* is a pointer to a buffer of size *length* for the result. The size of compressed name is returned or -1 if there was an error.

dn_comp compresses the domain name *exp_dn* and stores it in *comp_dn*. The size of the compressed name is returned or -1 if there were errors. *length* is the size of the array pointed to by *comp_dn*. *dnptrs* is a list of pointers to previously compressed names in the current message. The first pointer points to to the beginning of the message, and the list ends with NULL. *lastdnptr* is a pointer to the end of the array pointed to *dnptrs*. A side effect is to update the list of pointers for labels inserted into the message by **dn_comp** as the name is compressed. If *dnptr* is NULL, names are not compressed. If *lastdnptr* is

NULL, the list of labels is not updated.

FILES

/etc/resolv.conf	see resolver(3N)

NAME

rcmd, rresvport, ruserok – routines for returning a stream to a remote command

SYNOPSIS

rem = rcmd(ahost, inport, locuser, remuser, cmd, fd2p);
char **ahost;
int inport;
char *locuser, *remuser, *cmd;
int *fd2p;

s = rresvport(port);
int *port;

ruserok(rhost, superuser, ruser, luser);
char *rhost;
int superuser;
char *ruser, *luser;

DESCRIPTION

rcmd is a routine used by the superuser to execute a command on a remote machine using an authentication scheme based on reserved port numbers. **rresvport** is a routine which returns a descriptor to a tli endpoint with an address in the privileged port space. **ruserok** is a routine used by servers to authenticate clients requesting service with **rcmd**. All three functions are present in the same file and are used by the **reshd**(1M) server (among others).

rcmd looks up the host **ahost* using **gethostbyname**(3N), returning –1 if the host does not exist. Otherwise, **ahost* is set to the standard name of the host, and a connection is established to a server residing at the well known Internet port *inport*.

If the connection succeeds, an endpoint is returned to the caller and given to the remote command as **stdin** and **stdout** with the tirdwr streams module pushed, enabling the UNIX read/write system calls to be performed. If the caller wishes to use the TLI library calls instead, the **rncmd()** version of this function should be called. If *fd2p* is nonzero, an auxiliary channel to a control process is set up, and a descriptor for it is placed in **fd2p*. The control process returns diagnostic output from the command (unit 2) on this channel and will also accept bytes on this channel as being UNIX signal numbers, to be forwarded to the process group of the command. If *fd2p* is zero, the **stderr** (unit 2 of the remote command) is made the same as the **stdout**, and no provision is made for sending arbitrary signals to the remote process. You may, however, be able to get its attention by using out-of-band data.

The protocol is described in detail in **reshd**(1M).

The **rresvport** routine is used to obtain a tli endpoint with a privileged address bound to it. This tli endpoint is suitable for use by **rcmd** and several other routines. Privileged Internet ports are those in the range 0 to 1023. Only the superuser is allowed to bind an address of this sort to a tli endpoint.

ruserok takes a remote host's name, as returned by a **gethostbyaddr**(3N) routine, two user names and a flag indicating whether the local user's name is that of the superuser. It then checks the files */etc/hosts.equiv* and, possibly, *.rhosts* in the local user's home directory to see if the request for service is allowed. A zero is returned if the machine name is listed in the *hosts.equiv* file, or the host and remote user name are found in the *.rhosts* file; otherwise, **ruserok** returns –1. If the superuser flag is 1, the checking of the *hosts.equiv* file is bypassed. If the local domain (as obtained from **gethostname** (2)) is the same as the remote domain, only the machine name need be specified.

SEE ALSO

rlogin(1C), resh(1C), intro(2), rexec(3X), rexecd(1M), rlogind(1M), reshd(1M)

DIAGNOSTICS

rcmd returns a valid tli endpoint descriptor on success. It returns –1 on error and prints a diagnostic message on the standard error.

rresvport returns a valid, bound tli endpoint descriptor on success. It returns –1 on error with the global value *errno* set according to the reason for failure. The error code EAGAIN is overloaded to mean “All network ports in use.”

NAME

rexec – return stream to a remote command

SYNOPSIS

rem = rexec(ahost, inport, user, passwd, cmd, fd2p);
char **ahost;
int inport;
char *user, *passwd, *cmd;
int *fd2p;

DESCRIPTION

rexec looks up the host **ahost* using **gethostbyname**(3N), returning –1 if the host does not exist. Otherwise, **ahost* is set to the standard name of the host. If a username and password are both specified, these are used to authenticate to the foreign host; otherwise, the environment and then the user's *.netrc* file in his home directory are searched for appropriate information. If all this fails, the user is prompted for the information.

The port *inport* specifies which well known DARPA Internet port to use for the connection; the call "getservbyname("exec," "tcp")" (see **getservent**(3N)) will return a pointer to a structure, which contains the necessary port. The protocol for connection is described in detail in **rexecd**(1M).

If the connection succeeds, a tli endpoint in the Internet domain of type SOCK_STREAM is returned to the caller and given to the remote command as **stdin** and **stdout**. If *fd2p* is nonzero, an auxiliary channel to a control process is set up, and a descriptor for it is placed in **fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel and will also accept bytes on this channel as being UNIX signal numbers, to be forwarded to the process group of the command. The diagnostic information returned does not include remote authorization failure, as the secondary connection is set up after authorization has been verified. If *fd2p* is zero, the **stderr** (unit 2 of the remote command) will be made the same as the **stdout**, and no provision is made for sending arbitrary signals to the remote process; you may, however, be able to get its attention by using out-of-band data.

SEE ALSO

rcmd(3X), rexecd(1M)

NAME

hosts – host name data base

DESCRIPTION

The *hosts* file contains information regarding the known hosts on the DARPA Internet. For each host, a single line should be present with the following information:

official host name
Internet address
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts.

Network addresses are specified in the conventional dot (.) notation using the *inet_addr*() routine from the Internet address manipulation library, **inet**(3N). Host names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/hosts

SEE ALSO

gethostbyname(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

hosts.equiv – list of trusted hosts

SYNOPSIS

/etc/hosts.equiv

DESCRIPTION

hosts.equiv contains a list of trusted hosts. If a host makes a **rlogin**(1) or **resh**(1) request, and the initiator of the request is in */etc/passwd*, no further validity checking is done. In other words, **rlogin** does not prompt for a password, and **resh** completes successfully. Therefore, a remote user is equivalenced to a local user with the same user ID when the remote user is in *hosts.equiv*.

The format of *hosts.equiv* is a list of names, as in this example: host1 host2 +@group1 –@group2 A line consisting of a simple host name means that anyone logging in from that host is trusted. A line consisting of +@*group* means that all hosts in that network group are trusted. A line consisting of –@*group* means that hosts in that group are not trusted. Programs scan *hosts.equiv* linearly and stop at the first hit (either positive for hostname and +@ entries, or negative for –@ entries). A line consisting of a single + means that everyone is trusted.

The *.rhosts* file has the same format as *hosts.equiv*. When user *XXX* executes **rlogin** or **resh**, the *.rhosts* file from *XXX*'s home directory is conceptually concatenated onto the end of *hosts.equiv* for permission checking. However, –@ entries are not sticky. If a user is excluded by a minus entry from *hosts.equiv* but included in *.rhosts*, that user is considered trusted. In the special case when the user is root, then only the */.rhosts* file is checked.

It is also possible to have two entries (separated by a single space) on a line of these files. In this case, if the remote host is equivalenced by the first entry, the user named by the second entry is allowed to log in as anyone, that is, specify any name to the **–l** flag (provided that name is in the */etc/passwd* file, of course). Thus, sundown john allows *john* to log in from sundown as anyone. The usual usage would be to put this entry in the *.rhosts* file in the home directory for *bill*. Then *john* may log in as *bill* when coming from sundown. The second entry may be a netgroup, thus +@group1 +@group2 allows any user in *group2* coming from a host in *group1* to log in as anyone.

FILES

/etc/hosts.equiv

SEE ALSO

rlogin(1C), resh(1C)

NAME

networks – network name data base

DESCRIPTION

The *networks* file contains information regarding the known networks which comprise the DARPA Internet. For each network, a single line should be present with the following information:

official network name
network number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network data base maintained at the Network Information Control Center (NIC) though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network number may be specified in the conventional "." notation using the **inet_network**() routine from the Internet address manipulation library, **inet**(3N). Network names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/networks

SEE ALSO

getnetent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

resolver – resolver configuration file

SYNOPSIS

/etc/resolv.conf

DESCRIPTION

The resolver configuration file contains information that is read by the **resolver** routines the first time they are invoked by a process. The file is designed to be human-readable and contains a list of name-value pairs that provide various types of resolver information.

On a normally configured system, this file should not be necessary. The only name server to be queried will be on the local machine, and the domain name is retrieved from the system.

The different configuration options are:

nameserver
followed by the Internet address (in dot notation) of a name server that the resolver should query. At least one name server should be listed. Up to MAXNS (currently 3) name servers may be listed; in that case, the resolver library queries tries them in the order listed. If no **nameserver** entries are present, the default is to use the name server on the local machine. (The algorithm used is to try a name server, and if the query times out, try the next, until out of name servers; repeat trying all the name servers until a maximum number of retries are made.)

domain followed by a domain name is the default domain to append to names that do not have a dot in them. If no **domain** entries are present, the domain returned by **gethostbyname** (3N) is used (everything after the first "."). Finally, if the host name does not contain a domain part, the root domain is assumed.

The name-value pair must appear on a single line, and the keyword (e.g., *nameserver*) must start the line. The value follows the keyword, separated by white space.

FILES

/etc/resolv.conf

SEE ALSO

gethostbyname(3N), resolver(3), named(1M)
Name Server Operations Guide for BIND

NAME

services – service name data base

DESCRIPTION

The *services* file contains information regarding the known services available in the DARPA Internet. For each service a single line should be present with the following information:

official service name
port number
protocol name
aliases

Items are separated by any number of blanks and/or tab characters. The port number and protocol name are considered a single *item*; a "/" is used to separate the port and protocol (e.g., 512/tcp). A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Service names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/services

SEE ALSO

getservent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

shells – shell database

DESCRIPTION

The *shells* file contains a list of the shells on the system. For each shell a single line should be present consisting of the shell's path, relative to root.

A "#" indicates the beginning of a comment; subsequent characters up to the end of the line are not interpreted by the routines which search the file. Blank lines are also ignored.

FILES

/etc/shells

SEE ALSO

passwd(1), ftpd(1M)

NAME

ip – Internet Protocol

DESCRIPTION

IP is the transport layer protocol used by internet based networks. IP is implemented as a Streams multiplexor providing the link between upper level protocols (TCP, UDP) and lower level network interfaces (i.e., Ethernet). IP supports packet routing, forwarding, packet fragmentation and reassembly, and the ICMP protocols.

IP maintains the description of all interfaces being used by the internet network system and may be retreived via the IP_GET_IF_CONFIG ioctl.

In addition, IP provides a raw ICMP interface to user level programs and a raw IP datagram facility.

NAME

TCP – Internet Transmission Control Protocol STREAMS driver

SYNOPSIS

```
#include <tiuser.h>
#include <netinet/in.h>

fd = open("/dev/tcp", oflag);
        or
fd = t_open("/dev/tcp", oflag, info)
int oflag;
struct t_info *info;
```

DESCRIPTION

The TCP protocol provides reliable, flow-controlled, two-way transmission of data. It is a byte stream protocol to support TLI connection oriented transport endpoints. TCP uses the standard Internet address format and provides a per-host collection of "port addresses." Thus, each address is composed of an Internet address.

Using open(2) directly on the TCP device opens a cloned tcp stream that will accept streams messages conforming to the TPI interface used by the TLI library. t_open automatically allows the TCP stream to be used from the TLI programming library.

Endpoints utilizing the tcp protocol are either "clients" or "servers." Client endpoints initiate connections to server endpoints. To create a server endpoint, the t_bind operation must be performed with the qlen parameter > 0. Unlike socket based interfaces, the t_listen operation only recieves connection indications it has no effect on the status of the endpoint. Only server endpoints may issue the t_accept call to complete the connection indication returned by a t_listen call. Client connections are generally initiated through the TLI t_connect library routine.

Passive endpoints may "underspecify" their location to match incoming connection requests from multiple networks. This technique, termed "wildcard addressing" allows a single server to provide service to clients on multiple networks. To create a port that listens on all networks, the Internet address INADDR_ANY must be bound. The TCP port may still be specified at this time; if the port is not specified, the system will assign one. Once a connection has been established, the Internet address is fixed by the peer entity's location. The address assigned the endpoint is the address associated with the network interface through which packets are being trnasmitted and received. Normally this address corresponds to the peer entity's network.

TCP supports options through the tli t_optmgmt call or directly (when it would violate TLI state controls) by the TCP_SETOPT ioctl.

NAME

udp – User Datagram Protocol

SYNOPSIS

```
#include <tiuser.h>
#include <netinet/in.h>

fd = open("/dev/udp", oflag);
        or
fd = t_open("/dev/udp", oflag, info)
int oflag;
struct t_info *info;
```

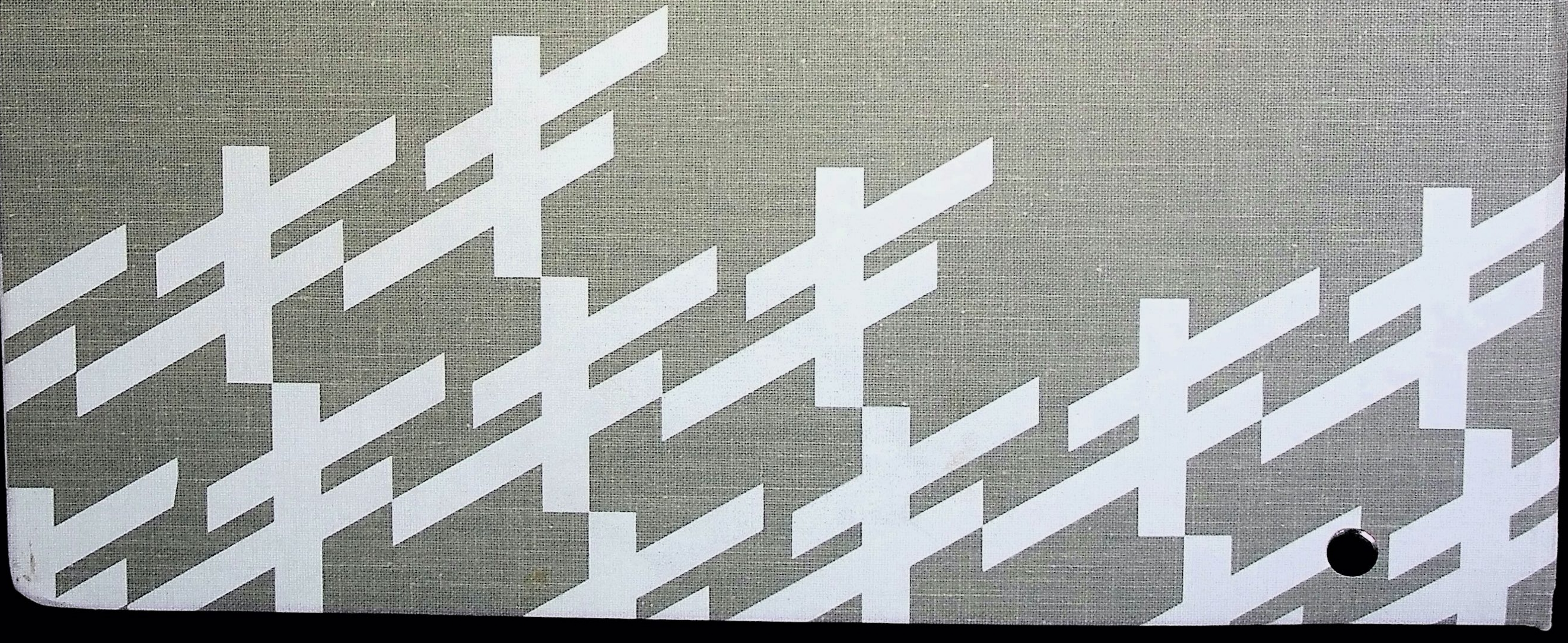
DESCRIPTION

UDP is a simple, unreliable datagram protocol which is used to support the connection-less (CLTS) transport service provided by the TLI interface. UDP endpoints are connectionless, and are normally used via the t_sndudata, t_rcvudata calls, although direct interface via streams messages may be performed.

UDP address formats are identical to thos used by TCP. In particular UDP provides a port identifier in addition to the normal Internet address format. Note that UDP port space is seperate from TCP port space. In addition, broadcast packets may be sent (if the underlying network supports it) by using the reserved broadcast address obtainable from IP. This address is network interface dependent.

The UDP streams multiplexor conforms to the TPI streams message interface defined for the TLI library packet. In addition, a set of ioctls are provided to set/retrieve UDP module configuration.

ptx/TCP/IP Release Notes

ptx/TCP/IP Release Notes