CTIX^{TM} INTERNETWORKING MANUAL (Preliminary Edition)

Preliminary Edition (May 1986) 09-00665-00

CONTENTS

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

1 RELATED DOCUMENTS

4.2BSD Networking Implementation Notes

4.2BSD Interprocess Communications Primer

4.2BSD System Manual

The C Programming Language, Kernighan and Ritchie

CTIX Operating System Manual, Version B

DDN MegaFrame Reference Manual

Internet Protocol Transition Workbook, SRI International

Internet Protocol - RFC-791

Transmission Control Protocol - RFC-793

Name, Addresses, Ports, and Routes, RFC-814

The Ethernet, A Local Area Network Data Link Layer and Physical Layer Specifications, (available from Digital Equipment, Intel, or Xerox corporations)

MegaFrame CTIX Administrator's Reference Manual

MegaFrame CTIX Operating System Manual

MightyFrame CTIX X.25 Network Gateway Manual

MightyFrame Administrator's Reference Manual

MiniFrame Administrator's Manual

Release Notice for MegaFrame CTIX TCP/IP

Release Notice for the MegaFrame CTIX X.25 Interface

Release Notice for MightyFrame CTIX TCP/IP

Release Notice for MightyFrame CTIX TCP/IP-X.25 Interface

Release Notice for MightyFrame CTIX X.25 Network Gateway

Release Notice for MiniFrame Ethernet CTIX TCP/IP

<u>X.25 Network Gateway Manual</u> (CTOS for MegaFrame)

CONVENTIONS

Underline
: Variables strings and the names of commands, when referenced in the text, are indicated by underlining, for example, the command, ftp. This convention is in place of italics used in UNIX manuals.

The names of manuals are also underlined.

Boldface
: Boldface strings are literals to be typed just as they appear, for example, a pathname, **/etc/rc**.

Boldface is also used for command options used within a command program, for example the **-v** option of ftp.

Boldface is also used for subcommands used within a command program, for example the **bye** command used within ftp.

References to CTIX Operating System Object Types

When first mentioned in a chapter, the names of CTIX commands, system calls, library routines, and files include a numeric reference designation to the appropriate CTIX Operating System Manual. For example, in the reference to the command, ftp(1N), the number in parentheses refers to the chapter number of the CTIX Operating System Manual which contains the description of the command. For more information, see the appropriate CTIX Operating System Manual.

ACKNOWLEDGEMENTS

CTIX Internetworking documentation is based in part on the Fourth Berkeley Software Distribution (BSD), under license from the Regents of the University of California.

Chapter 13, "Using the Programmatic Interface" is adapted from A 4.2BSD Interprocess Communications Primer, November 19, 1984, by Samuel J. Leffler, Robert S. Fabry, and William N. Joy, all from University of California, Berkeley.

1 INTRODUCTION

SCOPE OF THIS MANUAL

This manual provides a framework of information for understanding and using CTIX internetworking resources. These resources can be used at different levels depending on the type of network user. Accordingly the manual is divided into sections specifically aimed principally at the following types of users, or readers:

- the network user, who wishes to access resources located on networked machines in addition to his own. As a network user, you use specialized network commands, similar to other CTIX commands. To use this manual, you should have some familiarity with using standard CTIX commands, the shell, and basic utilities.

- the CTIX applications programmer, who requires interprocess communication to implement an application system across the network. Of course, a programmer is usually also a network user.

- the network administrator who is responsible for setting up and maintaining a network or internetwork. Of course, an administrator is usually also a network user.

Internetworking and the user types are explained further in this introduction.

TERMINOLOGY USED IN THIS INTRODUCTION

Acronyms and terms with special definitions in this manual are defined in the text where they occur. A glossary at the back of this manual defines some other selected terms. In addition there are some terms used in this introduction and the rest of the manual that you should be acquainted with before reading further.

4.3BSD. 4.3 Berkeley Software Distribution is the name of a UNIX standard developed and maintained by the Department of Electrical Engineering and Computer Science of the University of California, Berkeley. 4.3BSD is the designation of the 1986 release version. 4.2BSD is the 1983 release version.

CTIX-compatible. The ability of other UNIX systems (specifically 4.3BSD UNIX) to communicate with CTIX internetworking protocols and vice versa.

CTIX machine. A MightyFrame, MiniFrame, or MegaFrame or other computer that run the CTIX operating system.

DARPA. Department of Defense Advanced Research Project Agency, formerly called ARPA. This agency sponsored the network architecture research project upon which ARPANET is based. ARPANET is a large governmental internetwork, called the Internet, part of which is the Defense Data Network (DDN). See also **DDN and Internet.**

DDN. Defense Data Network. The Defense Data Network (DDN) is part of the DARPA Internet. The membership of the DDN is controlled by the Department of Defense (DoD). Members of the DDN are mostly government agencies, military, and universities engaged in defense-related work. The DDN allows users of these computer systems to send mail and files between systems and to access other computers on the network in interactive terminal sessions.

gateway. A software service installed at a switching node that connects two or more networks, especially if they use different protocols.

Internet. The Internet (spelled with initial capitalization) is the DARPA Internet System. See **DARPA.**

internetwork. An internetwork is made up of two or more networks. A CTIX internet is made up of two or more CTIX or CTIX-compatible nodes communicating over CTIX protocol(s).

layer. A layer is a separate set of clearly defined network functions that performs part of the communications tasks. Each layer interfaces only with its adjacent layers.

machine. A machine is a host computer. The use of this term is similar to "host," and "node," but "machine" connotates the machine-specific or hardware aspects of the host computer, whereas "node" connotates the logical aspects of a network host. Host connotates the relationship of the local node machine to application systems and remote hosts.

OVERVIEW

A network is two or more separate computer systems that provide their users with a common pool of resources by exchanging data over homogeneous communications links. An internetwork is two or more similar or dissimilar networks linked by communications gateways to form a supernetwork. Unless otherwise specified, this manual uses the term "network" or "networking" to include "internetwork" or "internetworking."

Networking communications requires software implemented on several levels, or layers, because of the complexity of the task. (A protocol is a set of conventions used to enable communicating entities to understand each other.)

The two most important levels from the point of view of this manual are the generalized terms "transport level protocol" and "link level protocol." In this manual, "protocol" signifies the network level and "media" signifies the media level. The media protocol in turn requires a specific set of communications hardware called the "physical media."

(For practical purposes, this manual generally uses "media" for both the plural "media" and the singular "medium.")

In a CTIX environment, CTIX Internetworking provides network communications among the MiniFrames, MightyFrames, and MegaFrames and other compatible UNIX systems.

Figure 1-1 diagrams a simple CTIX network. In this network, three CTIX machines and a compatible UNIX machine are linked via an Ethernet Local Area Network (LAN). The protocol used is Transmission Control Protocol/Internet Protocol (TCP/IP) and the media used is Ethernet. (These entities are explained in this introduction.)

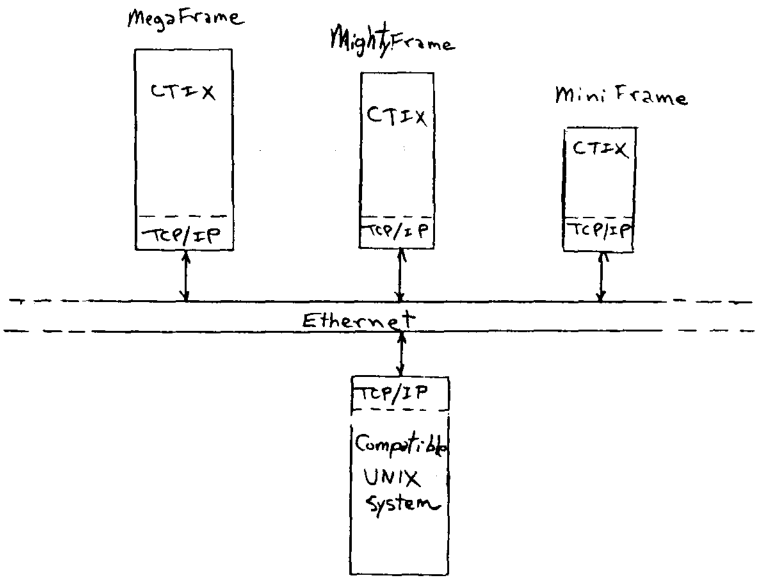


Figure 1-1. CTIX Internetworking Scheme Based on TCP/IP and Ethernet.

FEATURES OF CTIX INTERNETWORKING

CTIX Internetworking provides the usual advantages of internetworking.

- centralization of scarce resources
- avoids having to install and maintain duplicate services on multiple machines. For example, a communications server, such as the X.25 Network Gateway can be installed on only one MightyFrame node yet be accessible to others on the internet
- allows similar or dissimilar networks to be combined into a wide area internet

CTIX Internetworking supports file transfer, remote terminal, and electronic mail in a local area network environment. It provides the same services transparently through wide area leased line networks and public data networks (PDNs). CTIX Internetworking is characterized by these features.

- o supports multiple protocols and network media in various combinations

- o supports commonly used ARPANET and Berkeley 4.3BSD commands and is compatible with Berkeley 4.2BSD commands

- o complies with a wide range of DoD ARPANET specifications. (For a list of these, see the "Release Notice for MegaFrame TCP/IP.")

- o menu-driven <u>netman</u> for network users and administrators

- o provides for routing data to remote nodes through adjacent nodes

INTERNETWORKING PROTOCOLS

CTIX Internetworking is a protocol-independent networking system. Currently TCP/IP is the only supported transport/network protocol; however, other communications protocols can be added.

<u>Uucp</u>(1N) is a higher level protocol (presentation and session layers) that is fully supported by CTIX Internetworking. <u>Uucp</u> is a standard UNIX file transfer program that has protocol-like features such as reliable transport service and RS-232-C asynchronous autodial. It can run on top of TCP/IP in a CTIX internetwork based on Ethernet.

INTERNETWORKING MEDIA

CTIX Internetworking is an inherently media-independent networking system that can employ a number of link level media protocols. Currently the following media are provided:

- o Ethernet
- o X.25
- o Serial Line Internet Protocol (SLIP)

CTIX uucp can run on top of TCP/IP.

HOW TO USE THIS MANUAL

Network users, administrators, and programmers (as defined in "Scope of this Manual," above), can access the information they need by following the section-and-chapter organization of this manual. The manual is divided into five sections, each beginning with a printed tab divider.

The first chapter is each section contains suggestions for using the other chapters in the section.

Only Sections 1 and 2 are written for the network user. All other sections are designed primarily for network administrators and programmers.

Table 1-1 shows the section-and-chapter structure of the manual at a glance.

Table 1-1.

Section and Chapter Titles of the CTIX Internetworking Manual.

1	**CTIX Internetworking**
	Contents
	Related Documentation
	Ch 1, Introduction
2	**Using the Network**
	Ch 2, Using Network Commands
	Ch 3, Using the Virtual Terminal
	Ch 4, Transferring Files
	Ch 5, Using Netman
3	**Configuration and Management**
	Ch 6, Setting Up a Network
	Ch 7, Network Configuration
	Ch 8, Network Management
	Ch 9, Network Status Monitoring
	Ch 10, Internetworking Media
4	**Architecture and Programming**
	Ch 11, Internetworking Concepts
	Ch 12, System Architecture
	Ch 13, Using the Programmatic Interface
5	**Troubleshooting**
	Ch 14, Troubleshooting
6	**Appendixes**
	A, Telnet Command Summary
	B, FTP Command Summary
	C, /etc/rc
	D, /etc/hosts
	E, /etc/networks
	F, /etc/gethosts
	G, Internet Addresses
7	**Glossary**
8	**Index**

SECTION DESCRIPTIONS

The five manual sections are described as follows:

Tab 1	CTIX Internetworking. This section contains the manual contents, Related Documentation, and Chapter 1, "Introduction." All three types of readers should read this section.
Tab 2	Using the Network. This section contains Chapters 2 through 5. These chapters are written primarily for the network user. Chapter 2, "Using Network Commands," introduces the other three chapters. All network users should read Chapter 2. Chapters 3 through 5 go into the details of the networking commands used for common network user functions. You can use and reference these chapters as needed.
Tab 3	Configuration and Management. This section contains chapters 6 through 1Ø. They are intended for the network administrator. Chapter 6, "Setting Up a Network," introduces and serves as a guide for the section. The remaining sections can be referred to as needed from the master procedures in Chapter 6.

Tab 4	Architecture and Programming. This section contains Chapters 11 through 13. They are written primarily for the applications programmer; however, an administrator may find them useful as background for understanding and troubleshooting a network (especially Chapter 12, "System Architecture").
Tab 5	Troubleshooting. This tab section consists entirely of Chapter 14. It is provided for administrators, although programmers may find it helpful in debugging an applications system.
Tab 6	Appendixes. This section contains Appendixes A through F. Appendixes A and B are command summaries for telnet and ftp. These can be helpful to the network user. The remaining appendixes are for the administrator. They provide sample network database files.

The Glossary and Index, of course can be helpful to all three types of readers.

CHAPTER DESCRIPTIONS

This is both a tutorial and a reference manual. Its sections and chapters are organized in a certain logical sequence, but you can access sections, chapters, and even individual paragraphs randomly to accommodate your learning and re-reference pattern. "Signposts" are positioned at various transition points to help you go to the next step or find collateral or tangential material.

A paragraph at the beginning of each chapter summarizes the subjects treated in the chapter.

Chapter 2, "Using Network Commands," is an overview of using the network from the network user's point of view. It also leads into the following network user sections, which describe the major functions you can perform over the net.

Chapter 3, "Using the Virtual Terminal" explains in detail the virtual terminal commands, telnet and rlogin.

Chapter 4, "Transferring Files," explains how to use the file transfer commands, ftp and rcp.

Chapter 5, "Using Netman," describes the menu-driven user interface for the network.

Chapter 6, "Setting Up a Network," is an overall guide for the administrator in setting up and maintaining one or more nodes in a network.

Chapter 7, "Network Configuration," gives detailed information on some of the steps necessary to configure a node.

Chapter 8, "Network Management," tells how to load and maintain network services and provides some network management techniques.

Chapter 9, "Network Status Monitoring," describes the commands used to monitor network status and activities.

Chapter 10, "Internetworking Media," gives media-specific information on the media that are available for use with CTIX internetworking.

Chapter 11, "Internetworking Concepts" gives a survey of the major CTIX concepts and protocols.

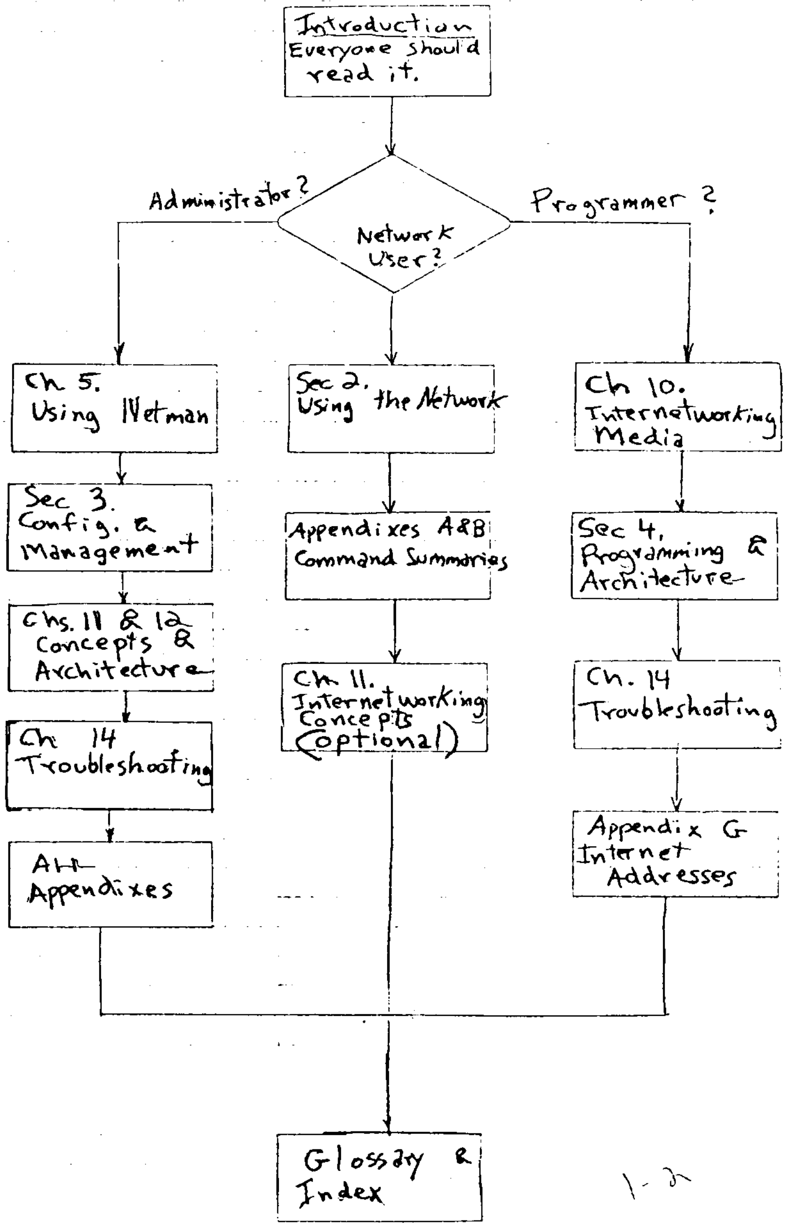
Chapter 12, "System Architecture," describes CTIX internetworking implementation and operation.

Chapter 13, "Using the Programmatic Interface," surveys the system call and library routines that the programmer can use in developing application systems for running over the net. This chapter also outlines a typical exchange between networked machines and provides other programming information.

Chapter 14, "Troubleshooting," provides some information to aid in maintaining a node or network and in remedying major networking problems.

WHAT'S NEXT?

The flowchart, Figure 1-2, gives suggested study paths for the three major types of readers of this manual. The flowchart directs you to each appropriate section or chapter in a recommended sequence. The first chapter in each section contains suggestions for using the other chapters in the section.


Introduction
Everyone should read it.
Administrator?
Network User?
Programmer?
Ch. 5. Using Netman
Sec 3. Config. & Management
Chs. 11 & 12 Concepts & Architecture
Ch 14 Troubleshooting
All Appendixes
Sec 2. Using the Network
Appendixes A&B Command Summaries
Ch. 11. Internetworking Concepts (optional)
Ch 10. Internetworking Media
Sec 4, Programming & Architecture
Ch. 14 Troubleshooting
Appendix G Internet Addresses
Glossary & Index

1-2

Figure 1-2. Manual Study Guide.

2 USING NETWORK COMMANDS

This chapter introduces this section of the manual, which is titled "Using the Network." This chapter is an overview of CTIX internetworking commands for the network user and for the new administrator or programmer. You can use this chapter as a guide to the other three chapters in this section.

Some of the subjects discussed in this chapter include:

- o CTIX network object types
- o network commands
- o use of a virtual terminal
- o remote command execution (rcmd)
- o transferring files to and from remote machines
- o remote printing
- o using pipes and shell scripts
- o user and machine equivalences and passwords

ABOUT THIS SECTION

Chapters 3 through 5 provide detailed explanations and examples of the networking commands used for common network user functions. You can use and reference these chapters as needed, but they are intended only to supplement the coverage of the commands found in the appropriate CTIX Operating System Manual. (See the note below.)

Chapter 3, "Using the Virtual Terminal" explains in detail the virtual terminal commands, <u>telnet</u> (1N) and <u>rlogin</u> (1N).

Chapter 4, "Transferring Files," explains how to use the file transfer commands, <u>ftp</u> (1N) and <u>rcp</u> (1N).

Chapter 5, "Using Netman," describes the menu-driven user interface for the network.

Table 2-1 references some of the things you may want to do over the internet to the chapters that describe them.

Table 2-1

Chapter Directory for Network Functions

Function	Command	Chapter
equivalences	<u>netman</u>	2, 5
remote command execution	<u>rcmd</u>	2
remote printing	<u>lp</u> (1)	2
shell scripts	<u>pipe</u> (2)	2
status	<u>netman</u>	5
transfer files	<u>ftp</u>, <u>rcp</u>	4
virtual terminal	<u>telnet</u>, <u>rlogin</u>	3

OVERVIEW

CTIX is a command-oriented operating system. To make use of the remote resources in a CTIX internetworking environment, the user invokes network-specific commands. These commands are fully integrated with CTIX and may be invoked from the shell command line and shell programs or executed from within user programs with the *fork*(2) or *exec*(2) system calls, or the *system*(3) library routine.

These commands are user processes of the operating system which require network software to function. In CTIX the name of the command is the same as the name of the file that contains the process program.

NOTE

The treatment of networking commands in this section (and the rest of the manual) is not intended to be the definitive reference source for network commands. The reference manual for all CTIX commands, system calls, library routines and file types is the appropriate *CTIX Operating System Manual*. (See "Related Documents.")

WHAT DOES INTERNETWORKING PROVIDE THE CTIX USER?

A CTIX network based on Ethernet provides a means of linking up to 1000 CTIX and UNIX machines so that the network user on any one of these machines can access resources and data on any of the other machines. A CTIX internet is two or more networks, possibly using a variety of machine types, protocols, and media, welded together in a flexible manner to form a larger network. The internetworking linkage is invisible at the command interface level so that the system appears to the network user as a single network.

Some of the many things you can do as a network user whose machine is connected in a CTIX network are as follows:

- Log onto another machine on which you have an account

- Move logically from one remote machine to another without having to enter your password (if your system administrators have "equated" the machines or if you have created a user equivalence for that machine)

- execute commands on any machine in the network

  - you can execute commands where the data is (thus avoiding the moving of files)

  - you can execute commands where the load is lowest

  - you can construct sequences of CTIX commands including pipes which move data between machines for processing. For example:

    ```
    cb main.c | pr -f | rcmd mifc lp
    ```

    where mifc is the name of the machine that has the printer you wish to use.

- Access public data from all machines
- Copy or transfer files from one machine to another
- Share remote devices such as printers and tape drives
- Share remote resources such as remote spoolers, network gateways to IBM hosts or X.25 PDNs
- Access electronic mail systems that have been implemented for the network
- Run applications resident on other machines
- Access other UNIX machines such as a VAX or Sun that are running the appropriate communications protocol.

CTIX NETWORKING OBJECT TYPES

There are five types of CTIX networking objects:

1. executable commands (See Table 2-2.)
2. demons, or servers, supporting the commands (See Chapter 8, "Network Management.")
3. configuration files (See Chapter 7, "Network Configuration.")
4. system calls (for use by programmers) (See Chapter 13, "Using the Programmatic Interface.")
5. library function calls (C programming language) (See Chapter 13, "Using the Programmatic Interface.")

All these types are documented in the first four chapters of the CTIX Operating System Manual. In that manual, all the CTIX programs, including the networking programs, are listed alphabetically within chapter types. The chapter number of the program is given in parenthesis after the name of the command. The networking commands are designated by a "N" (for network) or "MN" (maintenance networking) suffixed to the chapter number of the program. For example:

ftp(1N)

OVERVIEW OF CTIX NETWORKING COMMANDS

The networking commands only are listed alphabetically in Table 2-2 with a brief description. Not all CTIX networking commands are intended for use by the network user. Some are for network administrative functions.

Unless otherwise stated, all Berkeley-UNIX-type commands in this manual are compatible with both the 4.2BSD and 4.3BSD release versions.

Table 2-2

CTIX Networking Commands

Command	Description
enpstart(1NM)	(MightyFrame only) intelligent processor board startup
ftp(1N)	file transfer program
ifconfig(1NM)	configure network interface parameters
mkhosts(1NM)	make node name commands
netman(1NM)	form-based network management
netstat(1N)	show network status
rcmd(1N)	remote shell command execution ("rsh" in Berkeley UNIX)
rcp(1N)	remote file copy
rlogin(1N)	remote log in
route(1NM)	manually manipulate the routing table
ruptime(1N)	display status of nodes on local network
rwho(1N)	who is logged in on the local network
setaddr(1NM)	(MiniFrame only) set DARPA Internet address from nodename
setenet(1NM)	(MiniFrame only) write Ethernet address on disk
slattach, sldetach(1NM)	attach and detach serial lines as network interfaces
telnet(1N)	user interface to DARPA TELNET protocol
tftpd(1N)	user interface to the DARPA Trivial File Transfer Protocol (TFTP)
trpt(1NM)	print protocol trace
uucp(1C)	CTIX system to CTIX system copy

UNIX 4.3BSD Commands and ARPANET Commands

Included in the CTIX commands are a set of commands often referred to in a Berkeley UNIX environment as the "r-commands." The r stands for "remote." These commands work similarly to their Berkeley counterparts. These 4.3BSD type commands are designed to be UNIX-specific and are most suitably used when you are working on a Berkeley 4.3BSD type host. The Berkeley UNIX command "rsh", remote shell, is equivalent to the CTIX command, rcmd.)

Another set of commands, such as telnet and ftp, originated from ARPANET. They are designed to be operating-system independent. The protocols used in these commands are specified by the DoD Internet specification.

The major difference between these two different types of commands is that the 4.3BSD commands propagate UNIX-style permissions across the network. The ARPANET commands do not understand the UNIX permissions.

Using Networking Commands in Different Machine Environments

The CTIX Internetworking commands are invoked and execute the same way on all three machine products:

- o MightyFrame
- o MiniFrame and MiniFrame Plus
- o MegaFrame

However, in some earlier release levels of CTIX Internetworking protocols, certain commands may not be implemented on the MegaFrame or MiniFrame or may work somewhat differently. For a list of such commands, see the machine-appropriate CTIX Operating System Manual and the appropriate release notice for the protocol you are using. Usually, if the command is not implemented on the MegaFrame, a user message to that effect is displayed after you have entered the command. Where a command is used exclusively on a particular machine, this manual makes a note of it.

WHAT IS USER EQUIVALENCE?

User equivalence is an existing statement on a local machine to the effect that a particular user on a remote machine is equivalent to a user by the same or a different name on the local machine and has the exact same privileges as the existing local user. The equivalent user does not need a password to log in when (s)he uses a program that understands user equivalence. Implicit in this equivalence is that the remote user now has password privileges on the local machine.

The remote user still needs a separate account and password set up on the remote machine. The equivalent user can use the same name on both machines or a different name.

Note that you need to have an equivalence set up for your own user name even on your local machine. If you pipe to another machine (see sh(1) or rcp(1N), you will need an equivalence to that machine.

(For more information on equivalences, see Chapter 3, "Using Netman.")

VIRTUAL TERMINALS AND REMOTE LOGIN

The 4.3BSD command rlogin(1N) and the ARPANET command telnet(1N) provide the user with a virtual terminal capability. A virtual terminal is created when the user on one machine logs on to another machine and presents his terminal as being logically on that machine. Between CTIX-compatible machines, switching your terminal between machines can be as easy as typing the name of the machine to which you wish to connect.

Virtual terminal capability differs from remote command execution in that the user can use programs that depend on accessing the terminal directly, such as vi, netman, or rogue. These commands use the terminal in raw mode. That is, they read from the terminal character by character, instead of line by line.

The following is a brief overview of telnet and rlogin. For more information on these commands, see Chapter 3, "Using the Virtual Terminal."

TELNET(1N)

The _telnet_ command provides virtual terminal access to other machines on the internet. Using _telnet_, you can log in to any host on the network for which you have an account just as if you were a local user of that machine. Once _telnet_ is invoked, your terminal is linked to a remote machine and data that you type is passed to that machine. Responses from the remote machine will be displayed on your terminal's screen. (_Telnet_ ignores UNIX equivalences.)

For more information on _telnet_, see Chapter 3, "Using the Virtual Terminal."

REMOTE LOGIN (rlogin)

The virtual terminal command, _rlogin_, allows the user to remotely log into another CTIX-compatible machine. This command requires a password on the host you are logging into unless you have user equivalence on that machine. The command, _rlogin_, is a UNIX-specific command and is most suitably used when you are working on a Berkeley 4.3BSD type host or other CTIX-compatible host.

For more information on _rlogin_, see Chapter 3, "Using the Virtual Terminal."

TRANSFERRING FILES

FTP AND RCP

The ARPANET command, _ftp_, allows a user to manipulate files on two machines simultaneously. You can examine directories and move single or multiple files between systems. This program is designed to be highly independent of the operating system.

An additional feature of *ftp* is that it allows an anonymous user who does not have an account on your machine to pick up or deposit certain files without a password from a protected area of the *ftp* home directory. *Ftp* does not require (or understand) user equivalence.

The remote file copy command, *rcp*, does require user equivalence. The command, *rcp*, is a UNIX-specific command and is most suitably used when you are working on a Berkeley 4.3BSD-type host.

For more information of *ftp* and *rcp*, see Chapter 4, "Transferring Files."

UUCP(1C)

You can also use *uucp* to transfer files. If you use *uucp* to transfer files across a telephone line over the internet, it ties up the line for a single session until you release it.

Uucp is not strictly an internetworking command because it can be used without the internetworking protocol drivers. However it is logically and practically related to networking management. (See the discussion in Chapter 1, "Introduction.") See also *uuclean*(1M) and *uustat*(1C) in the machine-appropriate *CTIX Operating System Manual*.

Further information on *uucp* and instructions for Administrators who need to set up *uucp* to run over TCP/IP, in contained in Chapter 10, "Internetworking Media."

REMOTE COMMAND EXECUTION (rcmd)

The Berkeley UNIX command "rsh", remote shell, is equivalent to the CTIX command, rcmd(1N). The rcmd command allows you to send commands to remote CTIX machines for execution and have the results returned to you. To use rcmd you do not have to log onto the remote machine. (It is like a pipe to another machine.) This command is useful for constructing distributed shell programs. To use rcmd, the user must have equivalence on the target machine (the machine on which (s)he is trying to execute the command).

This command may only be used with remote machines running CTIX or a compatible operating system. Rcmd passes the command its standard input and outputs the command's standard output and standard error.

You must have **/usr/hosts** in your search path to access machines directly. (For more information, see the entry for rcmd(1N) in the appropriate CTIX Operating System Manual.)

INVOKING RCMD

Rcmd is invoked from the CTIX shell. You must specify the name of a remote machine and one or more commands to be executed, for example,

```
# rcmd admin <command>
```

In most cases, you may omit specifying rcmd to the shell and simply put the name of the remote machine and a command, for example,

```
# admin <command>
```

Your system administrator must have configured CTIX to accept the name of the remote machine without specifying rcmd in order for you to be able to use this feature. Your system administrator can advise you on how your machine is configured.

Also, you may specify two options when invoking rcmd:

-l user	Generally, the command you specify will be executed under your user name on the remote machine. This option allows you to specify that the command be executed under another user name, for example, **# rcmd admin -l tom <command>** Whether you use your user name or another user name, you must have established permission for yourself on the remote machine which will execute the command (using netman). The system administrator of the remote machine can advise you on how the remote machine is configured.
-n	This option prevents rcmd from sending the standard input file to the remote command you specify and prevents rcmd from reading up the standard input file by making its standard input **/dev/null** instead of rcmd's standard input. For example, **# rcmd admin -n -l tom <command>** "Reading up" means reading the file and buffering it. Rcmd buffers data in the standard input file regardless of whether the remote command reads it.

SAMPLE RCMD SESSION

The following example shows rcmd being used to run the who(1) command on a remote machine called "admin" and to place the output in a file on the local machine by redirecting standard output.

```
# rcmd admin who > /tmp/admin.who
```

REMOTE PRINTING

Remote printing (often referred to as remote spooling) is one of the services provided invisibly by internetworking facilities. As a network user, you need not learn any special networking commands to direct a file to a remote printer. When you wish to print to a remote printer, use the lp(1) command and specify the name of the printer as set up by your administrator. The administrator must have set up the remote printer as a "pseudo printer" on your local system.

If you are an administrator, you can find specific details and instructions on how to set up remote spooling in the administrator's manual for the appropriate machine. In general, you can set up a pseudo printer by writing a shellscript using uucp or rcmd. Uucp is recommended because it provides security in spooling. Rcmd requires a machine or user equivalence for the user lp.

SHELLSCRIPT PROGRAMMING USING THE INTERNET

Many useful shell programs can be written using the capabilities of the CTIX networking commands to use pipes across the network. (See pipe(2).) Such shell programs can be the glue that make a distributed system most useful. Some examples of systems based on shell programs are:

- remote line printer spooling using the System V _lp_ system.

- distributed text processing using _troff_(1). In this system, macroprocessing is done at the user's node, the font crunching is done on a lightly loaded back-end machine, and printing is done on a machine with a laser printer.

- a software distribution system using anonymous _ftp_ to load new software across the net. This feature can also tell the user what software is available and give detailed information about particular software packages.

- programs that back up file trees across the net.

- a program that automatically updates host files on all the machines in the network as new hosts are added. (See Chapter 7, "Network Configuration.")

CONNECTIONS, NAMES AND ADDRESSES

From the perspective of the user, internet protocols are connection-oriented. This means that for information to be communicated between your machine and a remote machine over the internet you must first have established a connection to that machine. Establishing a connection is similar to dialing a phone number when making a phone call; it defines the parties in the call and sets up a connection between them.

Although the data sent over the connection is packet-switched, rather than circuit switched as in the telephone system, the functions are alike. TCP performs the mechanics of establishing connections for you but in many cases, telnet and ftp in particular, you have to be aware of connections and give commands to get them established.

As with dialing a phone, you must first know how to reach the recipient of your call when setting up a connection. Each host on the internet has a unique address, like a phone number, by which it can be "called" in establishing a connection. Because network addresses are not always easy to remember, the internet software allows for the use of names instead of addresses. Host names are established by your system administrator who should tell you the names of the hosts with which you may communicate. Since hosts may be used for several purposes, it is possible to have several names (aliases) for the same host address. However, each name always stands for a single host address and will connect you to the same host each time you use it.

ACCESS AND PASSWORD PROBLEMS

Often in an internetworking environment, different host machines are under the jurisdiction of different departments and personnel. Those in charge of a host machine often wish to limit access to their own machine for various security and procedural reasons. Privileges to a machine can be given only from the machine in question. If you are unable to access a machine you have a need for, you or your supervisor can see the network administrator of the host machine you wish to access.

If you need access beyond anonymous ftp (see "Transferring Files" above), the administrator can set up a machine or user equivalence between your native host and the remote host. You will need the an account and password and on the remote machine. If you have an account on a remote machine, you can set up a user equivalence yourself. (See "What Is User Equivalence?" above.)

(For more information on equivalences, see Chapter 3, "Using Netman.")

3 USING THE VIRTUAL TERMINAL

This chapter explains two commands that provide virtual terminal capability. "Terminal" indicates that the command allows your terminal on your local machine to act as a terminal on a remote machine over the internet. "Virtual" indicates that no physical connection is made to the remote machine. Rather, the command simulates a physical line between your terminal and a remote machine.

The virtual terminal commands are

o *telnet*(1N)

o *rlogin*(1N)

(The virtual terminal capability does not include multiplexing and reliability.)

The *telnet* command provides virtual terminal access to other machines on the internet. Using *telnet*, you can login to any host on the network for which you have permission just as if you were a local user of that machine. Once *telnet* is invoked, your terminal is linked to a remote machine and data that you type is passed to that machine. Responses from the remote machine will be displayed on your terminal's screen.

For communicating with other machines running the CTIX operating system, the *rlogin* command can be used in place of *telnet*. *Rlogin* provides a virtual terminal access to CTIX-like machines that is specific to the CTIX operating system. See "The Rlogin Command," below.

THE TELNET COMMAND

Telnet is an interactive program which allows you to communicate with a remote machine in a terminal session. Once you invoke telnet, you will interact with telnet until you exit and return to the shell (calling program).

COMMAND AND INPUT MODES

Whenever telnet is connected to a remote machine, it operates in input mode. Input mode transfers all the characters you type to the remote machine and displays all data sent to you by the remote machine on your terminal's screen. The one exception to this is a special character called the escape character, ^], which places telnet in command mode if you type it. (This escape character is not the same as the **Escape** command of your keyboard. It is produced by typing **Control-]** or **Code-]**, depending on which keyboard you are using.)

In command mode, data that you type is interpreted by telnet to allow you to control telnet operation. Command mode is also active when telnet is not connected to a remote host.

TELNET OPTIONS

When telnet is in input mode, it communicates with the remote host based on a number of options. These options specify how operating system and terminal specific properties of terminal to computer communications, such as whether the echoing of the characters you type is done by telnet locally or by the remote machine, will be performed. Telnet and the remote machine you specify will negotiate these options and establish a compatible set of options for your terminal when you connect to a host.

INVOKING TELNET

You invoke telnet from the CTIX shell with the command telnet.

Optionally, you may specify the name of the remote machine with which you wish to communicate. For example:

```
$ telnet admin
```

Machine names are defined by your system administrator. Before using telnet, you can use the netman command to examine the machine names available to you. For a description of how to examine the machine names available to you, see Chapter 5, "Using Netman."

When you specify a machine name when you invoke telnet, telnet will establish a network connection to that machine and enter input mode. You may also invoke telnet without a machine name, for example:

```
$ telnet
```

If you do not specify a machine name, you must open a connection from within telnet using telnet's open command before you can log into a remote host. See "Telnet Commands" below.

TELNET COMMANDS

You may enter telnet commands whenever the telnet command mode prompt is displayed. The telnet command prompt looks like:

```
telnet>
```

Telnet will be in command mode if you are not connected to a remote machine or when you enter the escape character from input mode.

If command mode was not entered from input mode, telnet will generally remain in command mode and display the command mode prompt again after you enter each command. If you use the open command to establish a telnet connection to a remote machine, telnet will enter input mode.

If command mode is entered from input mode, telnet generally will return to input mode after processing your command. If you use the close command to close the remote host connection, telnet will remain in command mode after the command is processed. If you use the **quit** command, telnet will exit and return you to the calling program, usually the shell.

Each command you give to telnet in command mode must be followed by a **Return.** Telnet will not start a command until it receives a **Return** from you. If you make a mistake while typing a command, you may use the shell line editing commands erase (**Backspace**) and kill (**Cancel**) to edit the characters that you have typed.

Telnet command names are in boldface.

When entering a command, you do not have to enter the full command name, only enough characters to distinguish the command from other telnet commands. In each command description, the minimum number of characters you are required to enter are underlined in the command name. A telnet command summary is provided in Appendix A.

AO	This command causes telnet to tell the remote machine to abort sending any output that is in progress. This command is useful if the remote host is sending you data that you do not wish to see and you would like telnet to return to command mode on the remote machine. The only output aborted is that currently being sent, you may continue to communicate with the remote machine once the current output has been stopped.
AYT	This command causes telnet to send an "are you there?" message to the remote machine. The remote machine will send you a message back if it is active. This message is often simply causing the bell on your terminal to sound although it may be a string of text which is displayed on your terminal. This message is useful if the remote host has not responded to your input and you wish to see if it is inactive or just busy.

BREAK This command sends a message to the remote machine which has the same significance as pressing the **Break** key on your terminal would to your local machine. Since **Break** is implemented between a terminal and a local machine as a set of physical signals, rather than data, pressing the **Break** key on your terminal affects only the local machine and is not sent to the machine to which you are connected via telnet. You must use the **Break** command if you want to send a break indication to a remote machine.

EC This command sends the telnet erase character message to the remote machine. **EC** has the same meaning as the shell erase (backspace) command does on your local machine. Since different operating systems implement the erase character operation differently, you may have to use the **EC** command, rather than the shell erase character, when interacting with a remote machine. The shell erase character can be used in command mode since command mode's operation is local to your machine.

EL This command sends the telnet erase line message to the remote machine. **EL** has the same meaning as the shell kill (erase line) command does on your local machine. Since different operating systems implement the erase line operation differently, you may have to use the **EC** command, rather than the shell kill command, when interacting with a remote machine.

The shell kill command may be used in command mode since command mode's operation is local to your machine.

IP — This command sends the telnet interrupt process message to the remote machine. **IP** has the same meaning as the shell interrupt command does on your local machine. Since different operating systems implement the interrupt operation differently, you must use the **IP** command, rather than the shell interrupt command, when interacting with a remote machine. The shell interrupt command may be used in command mode since command mode's operation is local to your machine.

SYNCH — This command sends a message to the remote machine telling it to ignore any input you have sent but which has not yet been processed on the remote machine. This command is useful if you have typed ahead a number of commands and wish to cancel these commands without terminating the telnet connection to the remote machine.

close — This command closes the connection to the remote host and causes telnet to enter command mode. This command is useful if you wish to stop communicating with one machine in order to start a connection to a new machine without leaving telnet.

crmod

This command turns on carriage return mapping in which a carriage return character from the remote machine is turned into a carriage return character followed by a line feed character. The next time you enter this command, telnet will turn carriage return mapping off.

escape

This command allows you change the escape character used to enter command mode from input mode. After you enter this command, telnet prompts you for a new escape character to be used. Once you enter this character followed by a **Return**, the new escape character will be in effect.

help

This command displays information on your terminal about operating telnet. If you specify a command name to help, information about that command is displayed. If you just enter help, a list of all commands is displayed.

open

This command establishes a telnet connection to a remote machine. You may specify the name of the remote machine when invoking the command, for example,

```
telnet > open admin
```

or you may just use the command name and have telnet prompt you for the machine name, for example,

```
telnet> open

(to) admin
```

You may only use this command when you do not already have an open telnet connection.

options	This command turns on display of the messages sent between telnet and the remote machine to negotiate options. The next time you enter this command, telnet will turn option negotiation display off.
quit	This command terminates your telnet session and exits telnet. The quit command closes the connection to the remote machine if one is active.
status	This command shows you the status of the connection to the remote host as well as the current options and escape character.
?	Another name for the help command.

SAMPLE SESSIONS

A number of sample sessions are shown below which illustrate how telnet can be used in a variety of ways. Communications with a host named USAF-TC are shown.

Session #1

```
$ telnet USAF-TC
Trying...
Connected to USAF-TC
Escape character is '^]'
Megaframe CTIX (tm: Convergent Technologies) User Mode (USAF-TC)
login: tom
$ ls
netman passwd volcopy whodo
$ ^]
telnet> quit
Connection closed.
$
```

Description of Session #1

This is a simple session illustrating basic telnet use. Telnet is invoked with a host name and opens a connection to that host. Telnet displays "Trying..." to indicate it is trying to establish a connection and a message indicating it is connected when the connection is established. Telnet displays the current escape character. (There is no options status display.) At this point, telnet has established the connection to the remote machine and the remote machine displays its login prompt. The user then logs into the machine using the same procedures that would be used for a local terminal on that machine. The user does a listing of his directory on the remote machine. Having completed his work, the user then types the escape character and telnet enters command mode and displays the command mode prompt. The user enters the quit command and telnet closes the connection to the remote machine and returns to the local shell.

Session #2

```
$ telnet
telnet> status
No Connection.
Escape character is '^]'
local echo is off
telnet> open USAF-TC
Trying...
Connected to USAF-TC
Escape character is '^]'
Megaframe CTIX (tm: Convergent Technologies) User Mode (USAF-TC)
login: tom
$ ls
netman passwd volcopy whodo
$ exit
Connection closed by foreign host.
$
```

Description of Session

This session illustrates alternative ways to log in and out of a remote machine with telnet. Telnet is invoked without a machine name and enters command mode. The user does a status command and telnet indicates that no connection is established. The user then uses the telnet **open** command to establish a connection and place telnet into input mode. The user receives a login message from the remote system. The user then logs into the machine using the same procedures that would be used for a local terminal on that machine. Having completed his work, the user logs out of the remote machine. The remote machine then closes the connection. Telnet terminates automatically and returns to the local shell.

THE RLOGIN COMMAND

The rlogin(1N) command connects you to a shell on a remote machine. Rlogin is similar to telnet but is specific to CTIX-compatible machines and allows you to access the same CTIX commands on a remote machine as telnet but is more convenient than telnet in that, once you have logged onto a remote machine, it is as if it is now your local machine and you do not have to know the special commands used in telnet. This command can only be used with remote machines running CTIX or a compatible operating system. The TERM variable in the remote shell is set to the value you are using in your local shell.

Once invoked, rlogin will pass all data you input to the remote machine and display all output from that machine on your terminal's screen.

INVOKING RLOGIN

Rlogin is invoked from the CTIX shell. You must specify the name of a remote machine, for example,

```
$ rlogin admin
```

In most cases, you may omit specifying rlogin to the shell and simply put the name of the remote machine, for example,

```
$ admin
```

Your system administrator must have configured CTIX to accept the name of the remote machine without specifying rlogin in order for you to be able to use this feature. You must also have **/usr/hosts** in your search path. Your system administrator can advise you on how your machine is configured.

Rlogin Options

Also, you may specify two options when invoking rlogin.

-ec	The **-e** options causes rlogin to use the character c instead of tilde (~) as the escape character to enter when exiting rlogin, for example,

```
$ rlogin admin -e!
```

sets the exclamation point as the rlogin escape character.

-l <user>	The -l option (lower-case L) allows you to specify that you wish to be logged in under another user name, for example,

```
$ rlogin admin -l tom
```

(Generally, rlogin logs you in to the remote machine with the same user name as you are using on your local machine.)

Whether you use your user name or another user name, you must have established user equivalence for yourself on the remote machine to which you are logging in. The system administrator of the remote machine can advise you on how the remote machine is configured.

USING A TILDE IN THE TEXT

To send a line of input beginning with a tilde (~) to the remote machine, begin that line with another tilde (the escape character).

EXITING RLOGIN

To exit rlogin and return control to your local shell, type the escape character (the tilde) and a period (~.).

Simply exiting your remote shell also causes rlogin to return control to your local shell.

4 TRANSFERRING FILES

This chapter describes two command programs you can use to transfer files, _ftp_(1N) and _rcp_(1N). Information in this chapter includes:

- o when and why to use the commands, including sample sessions
- o how to invoke and exit the commands
- o how to use the command options
- o detailed descriptions of the commands you can use within the _ftp_ program
- o description of the _rcp_ program

OVERVIEW

The _ftp_ command allows you to transfer files between your current node and other machines on the internet. _Ftp_ is an ARPANET command program. _Ftp_ is an interactive program which allows you to input a variety of commands for file transmission and reception, and for examining and modifying file systems of machines on the network. Once you invoke _ftp_, you interact with _ftp_'s command mode until you exit _ftp_ and return to the calling program.

Once _ftp_ is invoked, a set of commands is provided for use within _ftp_. These are described below in alphabetical order under "FTP Commands."

Ftp is available under a wide range of operating systems. When communicating with machines running the CTIX operating system, the _rcp_ command can be used in place of _ftp_. _Rcp_ provides file transfer among CTIX machines that is specific to the CTIX operating system.

THE FTP COMMAND

COMPATIBILITY OF FTP COMMANDS WITH INTERNET SYSTEMS

In addition to ftp commands that use standard ftp protocol functions, a number of commands are provided that use optional ftp protocol functions that cannot be supported by all operating systems. These commands should be used only in communicating with machines running CTIX or a compatible operating system. The commands whose use should be restricted in this way are indicated in the command descriptions, below. When communicating with a remote machine that does not run CTIX, you should ask your system administrator before using them whether it supports these ftp commands. Some ftp server do not support all the commands.

FTP SESSIONS

Ftp requires that you open a connection over the internet to a remote machine before you use an ftp command that involves that machine. Ftp allows you to have multiple connections active simultaneously, although generally you may only issue commands that operate on a single connection. The multiple connection facility allows you to communicate with several remote machine within a single ftp session without having to log in and out of these machines every time you wish to change connections. The connection that ftp will use at any given time is called the current connection.

Ftp connections are maintained with ftp's **open** and **close** commands. A connection created with the **open** command becomes the current connection and is used for subsequent file transfer commands until the next **open** command you issue. See "Invoking FTP" for a description of the **open** and **close** commands.

Generally, you will close a connection with the **close** command before you use the **open** command to begin a new connection. However, if you will be communicating with several hosts during an ftp session, you may wish to have several connections to different hosts open at the same time. To accomplish this, do not use the **close** command on your current connection before using **open** to begin a new connection.

Although the **open** command will cause the new connection you request to become the current connection, your old connection will not be closed. To switch among several active connections, use the **open** command with the host name you used to open one of your connections originally. This will cause the connection you previously opened to become the current connection rather than starting a new connection. Also, you will need to have several connections open to use the **copy** command. **Copy** requires that you do two opens without a **close** to establish connections to both the machine to copy from and the machine to copy to.

To close a previously opened connection other than the current connection, first use the **open** command to make the connection current and then close it with the **close** command. The **bye** and **quit** commands will close all your open connections for you.

FTP FILE TRANSFER MODES

Ftp allows you to transfer files in one of two modes: ASCII mode and binary mode. ASCII mode is used for text files which can be represented in standard ASCII code. Binary mode is used for binary data which must be represented as strings of contiguous bits. For communication among CTIX machines, the ASCII mode can be used for most file transfers. For communication to non-CTIX machines, the binary mode may be required for transferring some files such as program object modules. Your system administrator can advise you on when to use which file transfer mode.

FTP FILE NAMING CONVENTIONS

If the first character of a file name you specify to ftp is a hyphen (-), ftp uses its standard input (for reading) or the standard output (for writing).

If the first character of a file name you specify to ftp is a vertical bar (|), the remainder of the file name is interpreted as a shell command. Ftp will create a shell with the file name supplied as a command and then use its standard input (for reading) or the standard output (for writing). If the shell command includes spaces, the file name must be appropriately quoted. For example

```
"| ls -ls"
```

INVOKING FTP

You invoke ftp from the CTIX shell with the command ftp.

After ftp is started, the ftp prompt is displayed on your terminal. The ftp prompt looks like:

```
ftp>
```

Optionally, you may specify the name of the remote machine with which you wish to communicate. For example:

```
# ftp admin
```

Machine names are defined by your system administrator. Before using ftp, you can use the netman command to examine the machine names available to you. See Chapter 5, "Using Netman" for a description of how to examine the machine names available to you.

When you specify a machine name when you invoke ftp, ftp will establish a network connection to that machine to allow you to transfer files. This is equivalent to using the ftp open command to start a connection to the host you name. You may also invoke ftp without a machine name, for example:

```
# ftp
```

If you do not specify a machine name from the shell, you must open a connection from within ftp using ftp's open command before you can transfer files. See "FTP Commands" for a description of the open command.

Ftp Command Options

In addition to specifying a host name when invoking ftp, you may also specify a number of options which modify how ftp will operate. These options must be placed after the command name (ftp) but before the host name if you are specifying one. The options you may specify when invoking ftp all consist of a hyphen (-) followed by a single letter, for example, **-v**.

(Each of these options has a *corresponding* command, of the same name, that can be used within ftp. You should compare the use of the options with the corresponding ftp command. See "Ftp Commands" below, for a description of the ftp commands.)

-v causes ftp to operate in verbose mode. In verbose mode, the ftp protocol messages sent by the remote machine to ftp are displayed on your terminal. Also, if you use verbose mode, statistics are displayed after the completion of each file transfer. If you do not use the **-v** option, this information is not displayed. You may also modify whether verbose mode information is displayed from within ftp with ftp's verbose command.

-d causes ftp to operate in debug mode. In debug mode, the ftp protocol messages sent by ftp to the remote machine are displayed on your terminal. If you do not use the -d option, this information is not displayed. You may also modify whether debug mode information is displayed from within ftp with ftp's debug command.

-i causes _ftp_ to transfer files in image (binary) mode. If you do not use the **-i** option, files are transferred in ASCII mode. You may also modify which file transfer mode to use from within _ftp_ with _ftp_'s ASCII and binary commands.

-n causes _ftp_ to not use autologin mode when connecting to a remote machine. When autologin mode is used, _ftp_ will try to automatically identify you to the remote machine and log you in to that machine. If you use the **-n** option to turn off autologin, you will have to use the user command to login to the remote machine manually.

-g causes _ftp_ to disable expansion of CTIX file name wild cards such as *. If you do not use the **-g** option, _ftp_ will expand file names you enter with wild cards in them into lists of files. You may also modify whether wild card expansion is used from within _ftp_ with _ftp_'s **glob** command.

Some examples of options:

```
# ftp -v -d admin
```

invokes _ftp_ with verbose and debug modes on and causes _ftp_ to open a connection to the remote machine named admin. Debug mode causes the commands sent to the remote machine to be displayed. Verbose mode causes us to see the responses received and the statistics in bytes received.

# ftp -v -d

invokes ftp with verbose and debug modes on but does not cause any connection to be opened.

# ftp -n -g admin

invokes ftp with autologin and wild card expansion mode off and causes **ftp** to open a connection to the remote machine named admin.

# ftp -n -g

invokes ftp with autologin and wild card expansion mode off but does not cause any connection to be opened.

Using the .Netrc File For Automatic Login

As an optional convenience feature, you can create a file named, **$HOME/.netrc** (4N), (in your home directory). This file contains a line entry containing the login data for each machine you wish ftp to open automatically.

When you invoke ftp specifying a machine, or when you subsequently open a machine, ftp reads the **.netrc** file. If you have an entry for that particular machine, ftp automatically conducts the login protocol exchange with its counterpart at the remote machine. It supplies your login name and password if you have entered your password in the file. If you open a machine in verbose mode, you can see the transactions taking place.

The format of the file consists of blank-separated fields introduced by keywords:

```
 machine <name> login <name> password <password>
```

where machine, login, and password are keywords followed by the literal data needed for login:

machine	The name of the node.
login	The user login name for that node.
password	The user's password on that node. (The password is given in normal, unencrypted text.) If you do put your password in the file, you must read/write protect the file, by setting permissions, to prevent discovery of your password, otherwise ftp will not let you use the file. (There is still some risk here in putting your password in the file. You must weigh the security considerations.) Ask your system administrator before using this feature. If you do not enter your password in the file, ftp prompts you for your password.

Example:

```
    machine admin login superuser password open
```

where "admin" is the node; "superuser" is the user who logs into admin; "open" is superuser's password.

FTP COMMAND DESCRIPTIONS

When ftp displays this prompt, you can enter one of the commands described below. When the command is complete, the ftp prompt is displayed again. Depending on whether you turn on verbose or debug modes, other messages may also appear on your terminal.

Each command you give to ftp must be followed by a **Return.** Ftp will not start a command until it receives a **Return** from you. If you make a mistake while typing a command, you may use the shell line editing commands erase (**Backspace**) and kill (**Erase Line**) to edit the characters that you have typed.

You do not have to enter the full command name, only enough characters to distinguish the command from other ftp commands. In each of the following command descriptions, the minimum number of characters you are required to enter are underlined in the command name at the beginning of the description.

Ftp command names are in boldface.

!. The **!** command causes ftp to be suspended and a shell on the local machine to be invoked on your terminal. Any character(s) you type after entering the exclamation point are executed as a command. You can return to ftp by pressing the **Finish** key. This returns all ftp options and remote machine connections in the same state as before you gave this command.

append. The **append** command causes ftp to add the contents of a local file to the end of a file on the remote machine to which you are currently connected. You may specify the files to be used when invoking the command, for example

```
          ftp> append localfile remotefile
```

or you may just use the command name and have ftp prompt you for the file names, for example,

```
                    ftp> append
                    (local-file) localfile
                    (remote-file) remotefile
```

When you use the append command, the remote machine you are connected to must be a machine running CTIX or a compatible operating system.

ASCII. The **ASCII** command causes ftp to transfer files in ASCII mode.

bell. The **bell** command causes ftp to sound the bell at your terminal after each file transfer is completed. The next time you enter the bell command, ftp will stop sounding the bell after file transfers.

binary. The **binary** command causes ftp to transfer files in binary mode.

bye. The **bye** command terminates your ftp session and exits ftp. The bye command closes all your open connections.

c. The **cd** command changes the directory that you are working in on the remote machine to a new directory name. You may specify the new directory name when invoking the command, for example,

```
ftp > cd /usr/bin
```

or you may just use the command name and have ftp prompt you for the new directory, for example,

```
ftp> cd
(remote-directory) /usr/bin
```

close. The **close** command closes the current connection.

copy. The **copy** command is used to transfer a file between two remote machines without first moving the file to your local machine. You may specify the host and file names when you invoke the command, for example,

```
   ftp> copy admin:payroll finance:adminspayroll
```

Or you may just use the command name and have ftp prompt you for the host and file names, for example,

```
ftp> copy
(host1:file) admin:payroll
(host2:file) finance:adminspayroll
```

In order to use the copy command, you must first have opened connections to both of the remote machines. Also, you must use the same file transfer type for both connections.

debug. The **debug** command turns on and off debug mode. If debug mode is on, messages sent by ftp to the remote machine are displayed on your terminal. If debug mode is off, this information is not displayed.

delete. The **delete** command deletes a file on the remote machine to which you are currently connected. You may specify the name of the file to be deleted when invoking the command, for example,

```
ftp> delete remotefile
```

or you may just use the command name and have ftp prompt you for the file name, for example,

```
ftp> delete
(remote-file) remotefile
```

dir. The **dir** command displays a detailed listing of the contents of a directory on the remote machine to which you are currently connected. (Compare **ls** below.) You can specify the name of the directory to be listed when invoking the command, for example,

```
ftp> dir /usr/local/bin
```

If you do not specify a directory name, the current working directory on the remote machine is listed.

You can also specify that the results of this command are placed in a file rather than displayed on your terminal by giving ftp a file name on your local machine in which to store the directory listing, for example,

```
ftp> dir /usr/local/bin printfile
```

You must specify a directory name with the printfile. If you want to list the current directory in a file called "printfile," use:

```
ftp> dir . printfile
```

where "." stands for the current directory.

form. The **form** command displays the file format used. Currently, only the nonprint format is supported.

get. The **get** command retrieves a file from the remote machine to which you are currently connected and stores it on your machine. You may specify the name of a file on the remote machine and a file name on your machine for the file to be stored in when you invoke the command, for example,

```
ftp> get remotefile localfile
```

Or you can simply specify the name of a file on the remote machine to retrieve the file to your local machine and give it the same name as the file on the remote machine,

```
ftp> get remotefile
```

Or you may just use the command name and have ftp prompt you for the file names to use, for example,

```
ftp> get
(remote-file) remotefile
(local-file) localfile
```

If you omit the local file name, the get command will create a file on your machine with the same name as the file on the remote machine.

glob. The **glob** command causes ftp to disable expansion of CTIX file name wild cards such as *. The next time you enter the **glob** command, wild card expansion will be reenabled. If wild card expansion is enabled, ftp will expand file names you enter with wild cards in them into lists of files.

hash. The **hash** command causes ftp to display a pound sign (#) after each block of data it sends to or receives from the remote host. The size of a data block is 1024 bytes. The next time you enter the hash command, ftp will stop displaying pound signs after each data block.

help. The **help** command displays information on your terminal about operating ftp. If you specify a command name to help, information about that command is displayed. If you just enter help, a list of all the ftp commands is displayed.

lcd. The **lcd** command changes the working directory used by ftp on your machine. You may specify a directory name to be used as the working directory, for example,

```
ftp> lcd /usr/andy
```

If you do not specify a directory name, your home directory will be used.

ls. The **ls** command displays an abbreviated listing of the contents of a directory on the remote machine to which you are currently connected. You may specify the name of the directory to be listed, for example,

```
ftp> ls /usr/local/bin
```

If you do not specify a directory name, the current working directory on the remote machine is listed.

You may also specify that the results of this command are placed in a file rather than displayed on your terminal by giving ftp a file name on your local machine in which to store the directory listing, for example,

```
ftp> ls /usr/local/bin printfile
```

You must specify a directory name with the printfile. If you want to list the current directory in file called "printfile," use:

```
ftp> ls . printfile
```

where "." stands for the current directory.

mdelete. The mdelete command deletes a list of files on the remote machine to which you are currently connected. You may specify the name of the files to be deleted when invoking the command, for example,

```
ftp> mdelete remotefile1 remotefile2
```

or you may just use the command name and have ftp prompt you for the file name(s), for example:

```
ftp> mdelete
(remote-file) remotefile1 remotefile2
```

mdir. The **mdir** command obtains a directory listing for a list of remote files and places the result in a local file. You may specify the list of remote files and the local file when invoking the command for example,

```
ftp> mdir remotefile1 remotefile2 printfile
```

or you may just use the command name and have ftp prompt you for the file name, for example,

```
ftp> mdir
(remote-files) remotefile1 remotefile2 printfile
local-file printfile? y
```

mget. The **mget** command retrieves several files from the remote machine to which you are currently connected and stores them on your machine. The files stored on your machine have the same names as the files on the remote machine.

You may specify the list of remote files when invoking the command for example,

```
ftp> mget remotefile1 remotefile2
```

or you may just use the command name and have ftp prompt you for the file names, for example,

```
ftp> mget
(remote-files) remotefile1 remotefile2
```

mkdir. The **mkdir** command creates a directory on the remote machine to which you are currently connected. You may specify the name of the directory to be created when invoking the command for example,

```
ftp> mkdir /u/mydir
```

or you may just use the command name and have _ftp_ prompt you for the directory name, for example,

```
ftp> mkdir
(directory-name) /u/mydir
```

Not all _ftp_ servers support the **mkdir** command.

mls. The **mls** command obtains an abbreviated directory listing for a list of remote files or directories and places the result in a local file. You may specify the list of remote files or directories and the local file when invoking the command for example,

```
ftp> mls remotefile1 remotefile2 printfile
```

or you may just use the command name and have _ftp_ prompt you for the file name, for example,

```
ftp> mls
(remote-files) remotefile1 remotefile2 printfile
local-file printfile? y
```

mput. The **mput** command transfers several files from the local machine to the remote machine to which you are currently connected and stores them. The files stored on the remote machine have the same names as the files on your machine.

You may specify the list of files when invoking the command for example,

```
ftp> mput localfile1 localfile2
```

or you may just use the command name and have _ftp_ prompt you for the file names, for example,

```
ftp> mput
(local-files) localfile1 localfile2
```

open. The **open** command establishes a connection to a remote machine which may then be used for file transfer commands. You may specify the name of the remote machine when invoking the command, for example,

```
ftp > open admin
```

or you may just use the command name and have *ftp* prompt you for the machine name, for example,

```
ftp> open
(to) admin
```

If you specify a host name when invoking the command, you may also optionally specify a port number on the remote machine. If a port number is specified, *ftp* will attempt to open a connection to the remote machine at that port rather than the default port for *ftp*. You should only use this option if you are asked to do so by your system administrator. If you do not specify a port number, *ftp* will not prompt you for one.

prompt. The **prompt** command causes *ftp* not to ask you for permission to proceed between files in multiple file commands such as **mget**. The next time you enter the prompt command, *ftp* will start asking you for permission to proceed between files.

put. The **put** command transfers a file from the local machine to the remote machine to which you are currently connected and stores it. You may specify the name of a file on your machine and a file name on the remote machine when you invoke the command, for example,

```
ftp> put localfile remotefile
```

or

```
ftp> put localfile
```

or you may just use the command name and have _ftp_ prompt you for the file name(s) to use, for example,

```
ftp> put
(local-file) localfile
(remote-file) remotefile
```

If you omit the remote file name, the put command will create a file on the remote machine with the same name as the file on the local machine.

pwd. The **pwd** command cause _ftp_ to print the name of the current working directory on the remote machine to which you are currently connected.

quit. (The same as the **bye** command above)

quote. The **quote** command causes the arguments you enter to be sent to the remote machine for execution. Arguments must be _ftp_ protocol commands and arguments. The _ftp_ protocol commands that a remote host supports may be displayed with the **remotehelp** command. You may enter the command string to be sent when invoking the command, for example,

```
ftp> quote NLST
```

or you may just use the command name and have _ftp_ prompt you for the command line to use, for example,

```
ftp> quote
(command line to send) NLST
```

You should not use this command unless asked to do so by your system administrator.

recv. (The same as the **get** command above)

remotehelp. The **remotehelp** command requests help from *ftp* at the remote machine to which you are currently connected. The information returned by the remote machine indicates which *ftp* commands it can support.

rename. The **rename** command renames a file on the remote machine to which you are currently connected. You may enter the file names to be used when invoking the command, for example,

```
        ftp> rename remotefile1 remotefile2
```

or you may just use the command name and have *ftp* prompt you for the file names to use, for example,

```
                ftp> rename
                (from-name) remotefile1
                (to-name) remotefile2
```

rmdir. The **rmdir** command removes a directory on the remote machine to which you are currently connected. You may specify the name of the directory to be removed when invoking the command for example,

```
                ftp> rmdir /u/mydir
```

or you may just use the command name and have *ftp* prompt you for the directory name, for example,

```
                ftp> rmdir
                (directory-name) /u/mydir
```

Not all *ftp* servers support the **rmdir** command.

send. (The same as the **put** command above)

sendport. The sendport command causes _ftp_ to disable specifying a local port to the remote machine for a data connection. The next time you enter the sendport command, specification of local ports will be reenabled. The default mode for local port specification when _ftp_ is invoked is on. You should not use this command unless asked to do so by your system administrator.

status. The **status** command causes _ftp_ to display its current status on your terminal. This status includes the modes you select with the **bell, form, hash, glob, port, prompt,** and **type** commands.

type. The **type** command sets the file transfer type to one that you specify. Valid values are ASCII and binary. The type command is another way of invoking the ASCII and binary commands. If you do not specify a type when invoking this command, ASCII is used.

trace. The **trace** command causes _ftp_ to enable packet tracing. The next time you enter the trace command, specification packet tracing will be disabled. You should not use this command unless asked to do so by your system administrator.

user. The **user** command allows you to identify yourself to the remote host when establishing a connection. If autologin was not disabled with the **-n** option when invoking _ftp_, this command is not required. If autologin is disabled or an autologin is not configured for you on the remote machine, you will have to use the **user** command to tell the remote machine who you are.

Three pieces of information are used to tell the remote machine who you are: a login name, a password, and an account name.

User name is required for all machines, password and account name are required only by some systems. Your system administrator can tell you what information is required by what machines and what are valid user and account names and passwords for a machine you wish to communicate with.

(See "Using the **.netrc** File For Automatic Login," above.")

You may enter the information to be used with the user command when invoking the command, for example,

```
ftp> user tom cat myaccount
```

Also you may just use the command name and have ftp prompt you for the information to use, for example,

```
ftp> user
(username) tom
password:
Account: myaccount
```

Note that ftp will not echo your password when you type it to protect the security of this information. If a password or account is not required on the remote machine with which you are connecting, the password and account prompts will not be displayed.

verbose. The **verbose** command causes ftp to enable verbose mode. The next time you enter the verbose command, verbose mode will be disabled. In verbose mode, the ftp protocol messages sent by the remote machine to ftp are displayed on your terminal. Also, if you use verbose mode, statistics are displayed after the completion of each file transfer. If you do not use verbose mode, this information is not displayed.

?. (Another name for the help command.)

SAMPLE FTP SESSIONS

These sample sessions illustrate how ftp can be used in a variety of ways. Three hosts are used in these sessions, the local host CT-SPG and the remote hosts USAF-TC and USAF-TC1.

Session 1

```
# ftp USAF-TC
ftp> cd /etc
ftp> ls
netman
passwd
volcopy
whodo
ftp> get passwd
ftp> put wall
ftp> ls
netman
passwd
volcopy
wall
whodo
ftp> bye
#
```

Description of Session 1

This is a simple session illustrating ftp use for sending and receiving files. Ftp is invoked with a host name and automatically logs the user into that host since the **-n** (disable autologin) option was not used.

The user first changes working directory on the remote machine to the /etc directory. Since neither the **-d** (debug) or **-v** (verbose) options were used, no messages other than the ftp prompt are displayed by ftp.

The user does a directory listing of the **/etc** directory on USAF-TC using the **ls** command for an abbreviated listing. Ftp shows four files in **/etc** on USAF-TC. The command get "passwd" is then issued to copy the file passwd from USAF-TC to CT-SPG. A file named passwd is created on CT-SPG since no local file name was specified.

The put command is then used to copy a file called wall from the current working directory on the local machine to the remote working directory (**/etc**) on the remote machine (USAF-TC). Once again, the same file name is used since no remote file name was specified. After the transfer is complete, a directory listing is requested which now shows five files in **/etc** on USAF-TC including the file wall which was just sent from CT-SPG.

The bye command is then used to exit ftp and return to the shell.

Session 2

```
# ftp USAF-TC
ftp> debug
Debugging on (debug = 1)
ftp> verbose
Verbose mode on.
ftp> cd /etc
---> CWD /etc
200 CWD command okay.
ftp> pwd
---> XPWD
251 "/etc" is the current directory.
ftp> hash
Hash mark printing on (1024 bytes/hash mark).
ftp> get wall myfile
---> PORT 3,20,0,2,4,51
200 PORT command okay.
---> RETR wall
150 Opening data connection for wall (3.20.0.2,1075)(24384 bytes).
#######################
226 Transfer complete.
24550 bytes received in 12.00 seconds (2 Kbytes/s)
ftp> close
---> QUIT
221 Goodbye.
ftp> bye
#
```

Description of Session 2

This session illustrates the displays caused by using a number of _ftp_ options. After invoking _ftp_ with the remote host name, the user issues commands to turn on debug and verbose mode. _Ftp_ displays messages indicating that these options are now enabled.

The user then changes the remote working directory to **/etc.** Since debug and verbose modes are on, _ftp_ displays messages showing the command sent to the remote machine, (---> CWD /etc), and the response received from the remote machine, (200 CWD command okay.). Note that the **cd** command, which has a form the same as CTIX's change directory command, is sent as a CWD command (for change working directory) to the remote machine. The CWD command is _ftp_'s way of saying **cd** independently of any specific operating system command language.

Following the **cd** command, the user does a **pwd** command to verify the working directory. Once again, _ftp_ displays the messages sent between the local and remote machines and then displays the current remote working directory. The user then turns on the **hash** option. _Ftp_ displays a message indicating that this option is now enabled.

The command "get wall myfile" tells _ftp_ to retrieve the file wall and place it in the file myfile in the user's local working directory. _Ftp_ displays the messages sent between the two hosts to begin the transfer and then prints a hash mark for each block (1024 bytes) of information received. After the transfer is complete, statistics are displayed showing the total time required and the data rate for the transfer.

After the file is received, the user closes the connection with the close command and exits ftp with the **bye** command.

Session 3

```
# ftp -v -d
ftp> cd /etc
Not connected.
ftp> open
(to) USAF-TC
Connected to USAF-TC
220 USAF-TC FTP server (Version 3.4 Mon Aug 12 16:48:21 PDT 1985) ready.
331 Password required for tom.
230 User tom logged in.
ftp> cd /etc
---> CWD /etc
ftp> ls
---> PORT 3,20,0,2,4,65
200 PORT Command okay.
---> NLST
150 Opening data connection for (3.20.0.2.1089) (0 bytes).
netman
passwd
volcopy
whodo
226 Transfer complete.
24 bytes received in 1.00 seconds (87.023 Kbytes/s)
ftp> open USAF-TC1
Connected to USAF-TC1
220 USAF-TC1 FTP server (Version 3.4 Mon Aug 12 16:48:21 PDT 1985) ready.
Name (USAF-TC1:tom): tom
Password:
331 Password required for tom.
230 User tom logged in.
ftp> cd /etc
---> CWD /etc
ftp> get wall
---> PORT 3,20,0,2,4,66
200 PORT command okay.
---> RETR wall
150 Opening data connection for wall (3.20.0.2,1090)(24384 bytes).
226 Transfer complete.
24550 bytes received in 12.00 seconds (2 Kbytes/s).
ftp> open USAF-TC
Re-connected to USAF-TC.
ftp> put wall
---> PORT 3,20,0,2,4,67
200 PORT command okay.
---> STOR wall
150 Opening data connection for wall (3.20.0.2,1091)(24384 bytes).
226 Transfer complete.
24550 bytes received in 12.00 seconds (2 Kbytes/s).
ftp> bye
---> QUIT
221 Goodbye.
---> QUIT
221 Goodbye.
#
```

Description of Session 3

This session illustrates the use of multiple connections to remote hosts. The user invokes _ftp_ with verbose and debug modes on but without a host name and then tries to do a **cd** command. Since no connection to a remote host is active, _ftp_ cannot execute this command and indicates that it is not connected.

The user then uses the open command to establish a connection to USAF-TC. Note that _ftp_ prompts for the remote machine name and logs the user into that machine automatically. The user then changes his/her working directory to /etc on USAF-TC and lists the contents of this directory. Noticing that the file he needs is not in the directory, the user then opens a connection to USAF-TC1.

Since the user did not have an automatic login configured for USAF-TC1, _ftp_ asks the user for his/her name and password. Since an account name is not required on USAF-TC1, _ftp_ does not ask for one. Once logged in on USAF-TC1, the user changes working directory to /etc and transfers the file wall to his local working directory. Then, the user opens a connection to USAF-TC. Since the user did not issue a close command before opening his connection to USAF-TC1, his connection to USAF-TC is still active and a message indicating that the user is reconnected to USAF-TC is displayed.

The user then transfers wall to USAF-TC. Since the connection to USAF-TC was never closed, the user does not have to issue another **cd /etc** command to set his working directory before sending the wall file to the remote machine. After completion of the transfer, the user issues a bye command and _ftp_ closes both of his/her open connections and returns to the shell.

CTIX FILE COPY - THE RCP COMMAND

The _rcp_ command allows you to copy files between any two CTIX machines on the internet. _Rcp_ is similar to _ftp_ but has a syntax much like the CTIX _cp_ command. This command may only be used with remote machines running CTIX or a compatible operating system.

INVOKING RCP

Rcp is invoked from the CTIX shell. You must specify the name of local files to copy and where they are to be copied to, for example,

```
# rcp localfile admin.tom:/usr/local/bin
```

As shown, file names for _rcp_ follow a convention that is an extension of the CTIX file name convention. File names may take one of three forms, where a file name names a file or a directory. Valid forms for file names are:

o   machine.user:filename

o   machine:filename

o   filename

where,

machine	is the name of the machine which contains or will contain the file. If you do not specify a machine, the file is assumed to reside on your local machine.

user	is the user name on the machine you specify. If you do not specify a user name, your user name on your local machine is used. Whether you use your user name or another user name, you must have established permission for yourself on the machine where the file is located. The system administrator of the remote machine can advise you on how the remote machine is configured.
filename	is a standard CTIX file name which may include a directory path. If the filename you specify does not begin with a slash (/), the filename is assumed to be relative to the specified user's home directory. The filename may include wild cards but these may have to be quoted to prevent their expansion on your local machine.

An exclamation point may be used in place of the colon in rcp filenames.

If you specify only a directory name for the destination of an rcp command, the file(s) you specify are copied into that directory with the same names as the files copied.

Options to rcp

You can specify an option when invoking rcp.

-r	This option allows the copying of directory trees. If the file specified for copying is a directory and you specify -r, the entire directory tree under that directory is copied. When -r is specified, the destination of the <u>rcp</u> command must be a directory. When you do not specify the -r option, requesting the copying of a directory is an error.

SAMPLE RCP SESSIONS

In the following examples, two remote machines on the network are used named USAF-TC and USAF-TC1.

The first example copies a file named list from the user's current directory to his/her home directory on USAF-TC:

```
# rcp list USAF-TC:list
```

The next example copies a directory hierarchy to a directory tree rooted in src within user's home directory on USAF-TC.

```
# rcp -r /net/src USAF-TC:src
```

This example shows the user copying a file from the home directory of a user named tom on USAF-TC to the **/usr/tmp** directory on USAF-TC1. The copy on USAF-TC1 is to belong to a user named Andy.

```
# rcp USAF-TC.tom:list USAF-TC1.andy:/usr/tmp
```

5 USING NETMAN

This chapter introduces the menu and forms-based network management tool, netman. This chapter describes only the network user's version. The administrator's version contains additional network management functions. (See Chapter 8, "Network Management.")

OVERVIEW

CTIX Internetworking provides a unique network management program, netman. The user version of netman, is a terminal-independent program which network users can access to view the certain network status information and to add and delete their remote user equivalences to and from their account. (A user equivalence gives their remote accounts privileges equivalent to their own or someone else's local account. This is explained in greater detail in this chapter.)

Netman is menu and forms-driven; the user does not need to know CTIX editors or other tools. Help files are accessible within all its forms.

The netman displays and interactive forms described in this chapter reflect a generic version of netman. The actual forms and displays may vary depending on the M/Frame system you are using.

INVOKING NETMAN

To call up netman, type the command:

```
netman
```

and press **Return**

The main menu appears as shown in Figure 5-1.

```
|--| Network Manager |------------------------------|
|                                                    |
|              Main Function Menu                    |
|                                                    |
|   * Machine Status                                 |
|     Network Users                                  |
|     Administration                                 |
|     Network Interface Statistics                   |
|                                                    |
|                                                    |
| Choose the network function you wish to perform    |
|____________________________________________________|
```

Figure 5-1. Netman Main Menu

OPERATING NETMAN

You can operate netman on a workstation, Programmable Terminal (PT), Graphics Terminal (GT), or other terminals. On terminals other than those supplied with M/Frame systems, the names and keyboard locations of the control keys can be different. Table 5-1 summarizes the netman key functions and gives some equivalent key functions for some terminals.

When the main menu is initially displayed, an asterisk (*) is shown at the left of the first line item, "Machine Status." This asterisk is the menu selection pointer. Use **Return** or the arrow keys to move the asterisk to the line selection you wish to access.

When you have selected a menu function with the asterisk, press **Next** to go to that display.

To return to the Main menu from a display you have selected, press **Cancel**. (On some terminals, press **Control-X**.)

Press **Next Page** to see additional page displays if any. If there are additional pages the word "-more-" is displayed in the lower right corner of the display. When there are no more page displays, the display returns to the main menu.

You can also use the down arrow key or **Return** to scroll one line at a time.

To exit a menu function and return to the main menu, press **Next**. (On some terminals, press **Line Feed**.)

Some displays are also interactive forms, to execute a completed form, press **Next**.

To exit from a display, press **Cancel**. (On some terminals, press **Control-X/Code-X**.)

You can press **Help** to display some helpful suggestions. (See "Help Displays," below.)

If you press the wrong key, the Valid Key menu is displayed. (See "Valid Key Display," below.)

Table 5-1

Netman Control Keys

Key	Function	Terminal Equivalent
Arrow Keys	Moves pointer to next entry.	
Bound	Calls up additional form.	
Cancel	Exits to main menu.	**Control-X**
Help	Displays help menu.	**<Escape>**
Next	Executes form and/or returns to main menu.	**<Linefeed>**
Next Page	Displays next page or returns to main menu.	
Return	Moves pointer to next entry.	
<wrong key>	Displays valid key menu.	

HELP DISPLAYS

You can press **Help** to display the help display associated with whichever menu or display is currently active. (On some terminals, press **<Escape>**). The help display for the Main menu gives a brief explanation of what you can do with netman and the basic instructions for operating it.

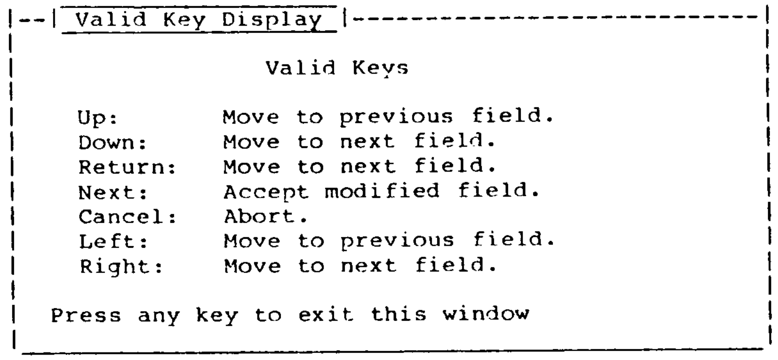
The file specification for the help file is given at the top of the display. For example:

```
Help (/usr/lib/netman/main.help)
```

Press **Next** to exit a help display.

VALID KEY DISPLAY

If you press an invalid key, the Valid Key Display for that particular display appears. The one shown in Figure 5-10. is displayed for the Main Function Menu display.



```
|--| Valid Key Display |----------------------------|
|                                                   |
|                   Valid Keys                      |
|                                                   |
|    Up:        Move to previous field.             |
|    Down:      Move to next field.                 |
|    Return:    Move to next field.                 |
|    Next:      Accept modified field.              |
|    Cancel:    Abort.                              |
|    Left:      Move to previous field.             |
|    Right:     Move to next field.                 |
|                                                   |
|  Press any key to exit this window                |
|___________________________________________________|
```

Figure 5-2. Valid Key Display.

MACHINE STATUS DISPLAY

Machine Status display is dynamically updated to report on current network machines. Machine Status shows the name of each machine active on the net, its current status, the time it has been up, its number of active users, and its recent work load. (This selection is equivalent to running the network command, ruptime.) Figure 5-3 shows an example display.

```
|--| Remote Machine Statistics |-----------------------------|
|                                                            |
| Machine     Status  Time      Users     Load               |
|                                                            |
| BTJmini      up      0:31,    1 user,   load 1.00, 1 00, 1 00 |
| CommDV       up     22:37,    1 user,   load 0.06, 0.06, 0.06 |
| TQmini       up     21:38,    1 user,   load 1.00, 1.00, 1.00 |
| Andrew       up   3+23:31,    1 user,   load 1.06, 1.06, 1.03 |
| dvl          up   6+18:22,    5 users,  load 1.00, 1.03, 1.27 |
| jjs          up   1+01:41,    0 users,  load 2.06, 2.06, 2.06 |
|                                                            |
|  Touch the 'next' key to continue                          |
|____________________________________________________________|
```

Figure 5-3. Machine Status Display.

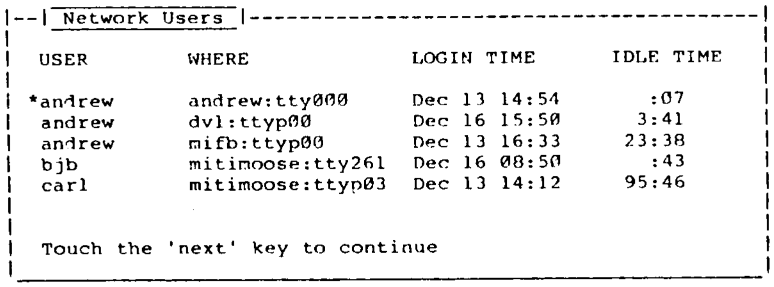
DESCRIPTIONS OF MACHINE STATUS DISPLAY HEADINGS

Machine	The name of the machine, or node.
Status	Whether the machine is up or down. A node is "down" if its <u>rwhod</u> server has not broadcast for five minutes. (See Chapter 8, "Network Management.")
Time	The time the node has been up in days, hours, and minutes. The plus sign (+) delimits the days from the hours.
Users	The number of logged-in users who have used their keyboards in the last hour.
Load	The average number of jobs in the run queue for the 1st one minute, five minutes, and 15 minutes.

NETWORK USERS STATUS DISPLAY

Network Users display is dynamically updated to report on logged in users. Itlists alphabetically all users currently logged onto the network. It displays their terminal session number and other information about their login session. (This selection is equivalent to running the network command, **rwho -a.**)

An example of the Network Users display is shown in Figure 5-4.



```
|--| Network Users |-----------------------------------------|
|                                                              |
|  USER          WHERE                LOGIN TIME      IDLE TIME |
|                                                              |
| *andrew        andrew:tty000        Dec 13 14:54       :07    |
|  andrew        dvl:ttyp00           Dec 16 15:50      3:41    |
|  andrew        mifb:ttyp00          Dec 13 16:33     23:38    |
|  bjb           mitimoose:tty261     Dec 16 08:50       :43    |
|  carl          mitimoose:ttyp03     Dec 13 14:12     95:46    |
|                                                              |
|                                                              |
|  Touch the 'next' key to continue                            |
|______________________________________________________________|
```

Figure 5-4. Network Users Status Display.

DESCRIPTIONS OF THE NETWORK USERS STATUS DISPLAY HEADINGS

USER	The name of each user who has been active during the last hour.
WHERE	The name of the user's machine and the terminal number attached to that machine.

	The "p" after tty signifies a pseudo terminal. A pseudo terminal is a virtual terminal created on the local node by the user from another node. A user can have more than one pseudo terminal. There is one for each virtual terminal session beyond the original session on the node where the user initially logged on.
LOGIN TIME	The date and time the user logged in for this session.
IDLE TIME	If the user has not used the keyboard in the last minute, the user's idle time is displayed here in hours and minutes. After 99:59 the value remains constant.

DETAILED USER STATUS DISPLAY

You can display additional information on what an individual user is doing by moving the asterisk down the left column to the name of the user and pressing **Next**. Figure 5-5 shows an example of the information displayed for the selected user.

```
|--| Detailed User Status |-----------------------------------------|
|                                                                   |
|  Process Information for user bjb at node mitimoose               |
|                                                                   |
|  UID        PID     PPID   C   STIME      TTY    TIME   COMMAND   |
|  bjb        10365    139   1   Dec 16     t261   0:04   /bin/ksh  |
|                                                                   |
|                                                                   |
|  Touch the 'next' key to continue                                 |
|___________________________________________________________________|
```

Figure 5-5. Detailed User Status Display.

Ordinarily you can check the Detailed User Status only for users on your local machine. To check the Detail User Status of a user on a remote machine requires a special permission. If you do not have this permission, the error message appears:

```
You don't have permission to check status
on machine xn
```

(Where "xn" is the name of the machine)

If you need this permission, see your administrator.

Descriptions of the Display Headings

UID	User ID.
PID	Process ID.
PPID	Parent Process ID.
C	Processor scheduling information.
STIME	Start Time.
TTY	Terminal Number.
TIME	CPU Run Time for the process.
COMMAND	The command the user is currently executing or has last executed.

For more information on process status reporting, see ps(1) in the appropriate CTIX Operating System Manual.

USER AND MACHINE EQUIVALENCE

(Administrators, see also Chapter 8, "Network Management.)

Equivalence can be established on a

- per user basis by making an entry in the file, **.rhosts**, on the local machine (This is called user equivalence.) You can make this entry by using an editor or by using netman.

- per machine basis (called machine equivalence) by entering the name of the remote machine in the file **/etc/hosts.equiv** on the local machine. (See your administrator for this type. For more information on machine equivalence, see Chapter 8, "Network Management.")

User equivalence is a mapping between a user on a remote machine and a user on a local machine. User equivalence allows a user from one machine to have user privileges on another machine under the same or different user name without entering a password for the remote machine.

This equivalence can be established by equating different users on different machines, on a host by host basis. For example, On host B, Joan, whose native host is host A, can be equated with Tricia of host B. See Figure 5-6.

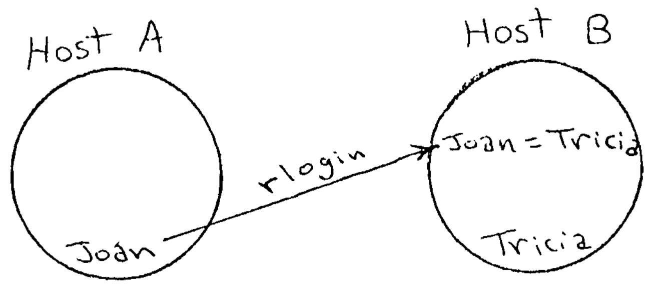


Figure 5-6. Diagram of an Example User Equivalence

User equivalence is established by the receiving machine, but the corresponding entry must be made in the **.rhosts** file on the user's own machine.

Because it it granted host by host, this type of equivalence is not reciprocal unless it is made explicit; that is, in the above user equivalence example, an equivalence set up on host B such that, Joan on host A = Tricia host B does not also mean that, on host A, Tricia B = Joan A, unless this reciprocal equivalence is set up separately.

User equivalence can also be made by equating the same user to him/herself on different machines. If the user desires a different name on a remote machine, (s)he must have a separate password entry for that name. In effect (s)he is a different user as far as the remote machine is concerned, but the different names can be equated using **.rhosts**.

User Equivalence to Self

It is recommended that a user be made equivalent to himself on his local machine. This equivalence is necessary to use the net under certain circumstances:

- o if you have shell script program that do a loopback operation (an operation in which a step in the program requires an operation on a remote host and then an operation back on your native machine)

- o if you run a command like Detailed User Status in netman the local machine

User equivalence is requireB for the Berkeley-derived utilities, rlogin, rcmd, and rcp. It does not work for standard ARPANET-derived utilities.

HOW TO SET UP USER EQUIVALENCES

As a network user, you can use Netman to set up user equivalences. On the Main Function Menu, select Administration. The Administration menu has two selections:

- o Add an equivalent user

- o Delete and equivalent user

Administration Menu

To call up netman's administration menu, select "Administration" on the Main menu. The User Administration menu appears as shown in Figure 5-7.

```
|--| User Administration |-----------------------------------|
|                                                             |
|                                                             |
|   * Add an equivalent user                                  |
|     Delete an equivalent user                               |
|                                                             |
| Move to the operation you wish to perform and touch `next`  |
|_____________________________________________________________|
```

Figure 5-7. User Administration Menu.

You use these menu functions only on your local machine. You can however establish a virtual terminal on the remote machine and invoke netman.

When you select a function on the Administration Menu, the corresponding form appears. After making an entry or editing the form as appropriate, press **Next** to execute the form.

If you need to edit your entries, use the cursor keys to position over the character you wish to change.

Add Equivalent User Form

When you select "Add an equivalent user," the form appears as shown in Figure 5-8.

```
|--| Add Equivalent User |-----------------------|
|                                                 |
|          Give a remote user your privileges     |
|                                                 |
|   Machine Name      |_______________|           |
|   Remote User                                   |
|                                                 |
|                                                 |
|   Use cursor keys to edit                       |
|_________________________________________________|
```

Figure 5-8. Add Equivalent User Form.

1. Enter the name of the remote machine to which you are granting the privileges.
2. Enter the name of the remote user to be granted the equivalence.
3. Press **Next** to execute the form.

Delete Equivalent User Form

When you select "Delete an equivalent user," the form appears as shown in Figure 5-9.

```
|--| Delete Equivalent User |--------------------|
|                                                 |
|     Retract your privileges from a remote user  |
|                                                 |
|   Machine/User      |_______________|           |
|                                                 |
|                                                 |
|   Move to your selection and touch 'next'       |
|_________________________________________________|
```

Figure 5-9. Delete Equivalent User Form.

1. The form comes up already displaying the name of one of the remote users. If this is not the name of the user you wish to delete, press **Bound** to get the Currently Equivalent User display as shown in the example in Figure 5-1Ø.

```
|--| Currently Equivalent User |-----------------|
|                                                |
|                Machine/User                    |
|                                                |
|  * tom-src   tom                               |
|    tom-src   tom                               |
|    mifa      tom                               |
|                                                |
|  Move to your selection and touch 'next'       |
|________________________________________________|
```

Figure 5-1Ø. Currently Equivalent Users Display

2. Using the arrow keys, move the asterisk to the machine/user name from whom you wish to retract the privileges and press **Next**. The name is inserted into the Delete Equivalent User form.

3. Press **Next** to execute the deletion.

6 SETTING UP A NETWORK

This chapter is an overall guide to configuring and managing a CTIX internetworking system. This Chapter also introduces tab section 3, "Configuration and Management," which consists of chapters 6 through 10. You can use this chapter as a starting point and refer back to it for the next step after performing specific procedures and information described in the release notices or other chapters. Table 6-1, at the end of this chapter, cross-references each area of network configuration and management to the chapter in this section that discusses it.

Chapters 7 through 10 discuss in greater detail specific subjects only touched upon in this chapter. This chapter discusses

- the administrator(s)

  system configuration of a node an network

  MightyFrame gateways

  routing

  overview of setting up a node

  release notices for machine types

- MightyFrame Setup Overview

- subjects covered in other chapters of this section

  the initialization files

THE ADMINISTRATOR

The administrator must bring each new machine into the network separately and locally. Similarly, once the local machine is on the network, the administrator typically maintains only his node's interface to the network. However, if there is a chief administrator or an administrator in charge of one or more machines, some maintenance tasks can be performed from a remote machine or the chief administrator can physically go around to each machine as necessary. There is no central network console.

Most configuration and network management tasks can be performed using netman.

CHIEF ADMINISTRATOR

In the case of larger networks, even if there is a chief administrator for a network, the administrator could be inhibited from performing some tasks by not having access to all the machine passwords.

ADMINISTRATOR PRIVILEGES

The administrator for each machine must have root privileges to accomplish his tasks. This means that you must know the password for the root account on your system and login to the system as root before you invoke netman to perform administrative duties.

SYSTEM CONFIGURATION

An internetwork consists of more than one physical network connected through common node(s). Machines that serve as connecting nodes are called gateways. Ethernet, SLIP connections, and X.25 links can be consolidated by gateways to form a transparently accessible internetwork. To the user an internetwork looks exactly like a larger single network. Some of the advantages of internetworking are:

- High speed local area networks (LANs) and slower wide area networks (WANs) can be mixed to allow for maximum balance between performance and economy. Typically, this means that people and machines within a local workgroup or LAN can get high performance networking among themselves and still have transparent access to remote people and machines.

- Gateways can allow high volume network traffic areas to be localized. On a large Ethernet, traffic from a small set of stations can congest the entire network. By localizing traffic efficiently through individual LANs connected by gateways, each part of the internet can have sufficient bandwidth.

- Previously existing networks can usually be combined into an internetwork by connecting a single gateway machine to the existing networks.

Figure 6-1 shows some of the ways an internetwork can be configured using the available media protocols and other connection protocols.

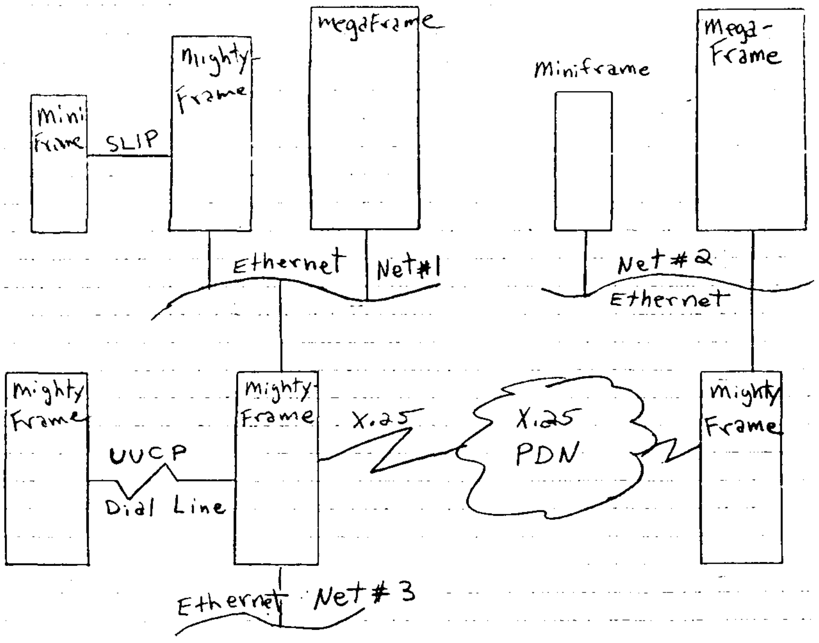


Figure 6-1. Example Internetwork Connected By Available Media

HOW ROUTING IS USED TO FORM AN INTERNETWORK

In order to use the internetworking features of CTIX TCP/IP, you must set up routes. A route is a record kept by the socket driver of which gateway to use in order to get to a certain remote network. These records are kept by the socket driver in a table. You can add and delete routes from the table using the route(1NM) command. Once a route to a remote network is established, users can use all the networking commands and refer to machines on that network as if they were on the local network.

For details on setting up routing, see Chapter 7, "Network Configuration."

EXAMPLE OF CONFIGURING AN INTERNET USING A GATEWAY MACHINE

Figure 6-2 shows how a MightyFrame can be used to connect an X.25 network and an Ethernet network.

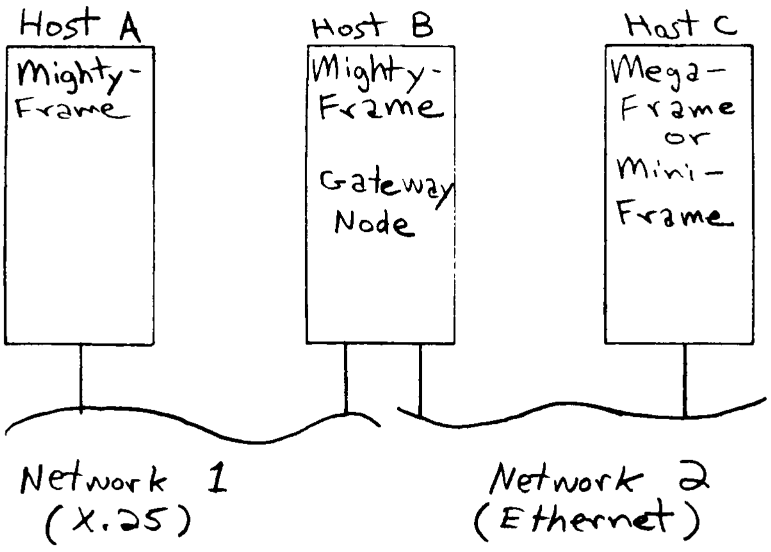


Figure 6-2. Example of An Internetwork Using a MightyFrame as a Gateway

For more information on system configuration, see Chapter 10. "Internetworking Media."

OVERVIEW OF SETTING UP A NEW NODE IN THE NETWORK

Some general aspects of the process of setting up a network node are presented here to give an overview of the detailed steps in the networking software release notice appropriate for your system.

Planning for network and internetwork topology and for the distribution and of network resources should be done with the entire, final configuration of the (inter)network in mind. The setup of the individual nodes should be coordinated with that of the other nodes to the extent this is practicable.

The process of setting up an individual network node on a specific machine type is detailed in the machine-appropriate release notice. The steps include the following categories, generally in the order given:

- configuring and installing the required hardware
- installing the required networking software from the distribution media to the system disk
- configuring the system files and initialization files for the higher level protocols networking software to be loaded
- configuring the system files and initialization files for the media protocols to be used in the system
- installing the loadable drivers containing the networking protocols by initializing the system (rebooting). (This step can be followed by the automatic downloading of the higher level protocol and/or media drivers to intelligent boards.)

- o initializing special media protocol such as SLIP
- o setting up (an possibly automating) the network database files
- o setting up routing paths
- o starting up network services

These steps are repeated for each node in the network.

The steps described above can are discussed further in this chapter according to the following organization:

- o preliminary steps (Follow these for all M/Frame machines.)
- o M/Frame-specific procedures (Select the appropriate one.)
  - MightyFrame
  - MiniFrame
  - MegaFrame
- o final steps to complete your node setup (Follow these for all machines.)

PRELIMINARY GUIDELINES IN SETTING UP A NODE

These guidelines are generalized to include all machines. They cannot substitute for the instructions in the machine-specific procedures of the appropriate release notice for the networking software. (See "Machine-Specific Procedures," below.

In addition, the release notices refer the appropriate administrator's manual for the particular system.

1. Check the current release notice to determine compatibility requirements between the internetworking protocol and the operating system version you are using.

2. Make sure that all hardware requirements for the data link and physical media are in place according to the directions in the appropriate release notice. (For more information on Ethernet hardware requirements, see also Chapter 1Ø, "Internetworking Media.")

3. Make sure the current networking software has been installed from the distribution media to the system disk, before you begin configuring the system.

MACHINE-SPECIFIC SETUP PROCEDURES

For the MightyFrame and MiniFrame, if you are merely updating your networking software, simply reboot the system after installing the software from the distribution media. If this is the first installation of TCP/IP on the machine, perform the procedures for adding a machine to a network as given in the appropriate release notice:

(Each machine has its own internetworking software package which is accompanied by a release notice.)

MightyFrame	Release Notice for MightyFrame CTIX TCP/IP
MiniFrame	Release Notice for MiniFrame Ethernet CTIX TCP/IP
MegaFrame	Release Notice for MegaFrame CTIX TCP/IP

MightyFrame Setup Overview

In the MightyFrame, the network interface for a given protocol is a loadable socket driver that runs in the kernel. (For more information on loadable device drivers and lddrv(1M), see the MightyFrame Administrator's Reference Manual.

MightyFrame runs the Ethernet driver on an intelligent controller board. The Ethernet driver portion of the TCP/IP loadable driver is downloaded onto the Ethernet controller.

MegaFrame Setup Overview

MegaFrame installation requires that both TCP/IP and the Ethernet driver be downloaded onto an intelligent board.

MegaFrame higher level networking protocols require a CTOS server in addition to the loadable protocol drivers.

AFTER SETTING UP YOUR MACHINE

After you have completed the appropriate machine-specific procedures, continue with the following administrative tasks to complete setting up the node.

1. The name of your machine must be included in the network databases contained on all nodes that intend to access your node. Your local node must also have an entry for all hosts it intends to access.

   If you have a large network or one that you expect to grow to a large network, setup a program to automate the updating of the network databases.

For details see Chapter 7, "Network Configuration."

2. When you have created the hosts database, run **/etc/mkhosts** to create the **/usr/hosts** directory. Users on your local machine need **/usr/hosts** in their search path to access machines directly. (For more information, see the entry for rcmd(1N) in the appropriate CTIX Operating System Manual.)

3. Set up appropriate user and machine equivalences, using netman. (For details, see Chapter 8, "Network Management.")

4. Start up the servers as explained in Chapter 8, "Network Management."

The machine is up and running in the network.

After your node is set up, monitoring status, additional network management tasks and techniques, and troubleshooting issues can be addressed. See Table 6-1, below.

WHAT'S NEXT

Table 6-1 cross-references each area of network configuration and management to the chapter in this section that discusses it.

Table 6-1

Chapter Directory for Administrator Tasks

Administrator Function	**Chapter**
adding hosts with netman	7
configuration files	7
database files	7
automatic updating	8
equivalences	8
Ethernet	1Ø
initialization file	6, 8
media, optional protocols	1Ø
netman	
configuring	7
managing	8
status monitoring	9
network names and addresses	7
routing	6, 7
services	8
SLIP	1Ø
status monitoring	9
system prompt	8
updating network software	8
uucp	1Ø

7 NETWORK CONFIGURATION

Section 6, "Setting up a Network" contains the basic procedures for adding a node to the network. This chapter and Chapter 8, "Network Management" give some background and more detailed descriptions for some of the steps necessary to configure a network node. This chapter contains

- o a discussion of network configuration files
- o a discussion of network names and addresses as background for understanding the process of establishing the network database files
- o recommendations and procedures for establishing and updating network database files
- o instructions for using <u>netman</u> to add or change a host entry in the **/etc/hosts** file
- o a discussion and instructions for setting up routing

CONFIGURATION FILES

Some of the files documented in Chapter 4 of the CTIX Operating System Manual are configuration files used in networking. These files are listed with brief descriptions in Table 7-1. All the files are designated (4N).

The administrator can modify these files to configure the node to local requirements. (The files **protocols** and **services** should not usually be modified.) These files can be manipulated directly using a text editor or with netman(1NM). All the following files are in the directory **/etc.**)

Table Table 7-1.

CTIX Networking Configuration Files.

File	Description
drvload(4N)	list of loadable drivers to be loaded at boot time
hosts(4N)	list of nodes on the network
networks(4N)	names and numbers for the network in the internet
protocols(4N)	list of internet protocols
services(4N)	list of internet services

In addition to the configuration files in Table 7-1, the file **.rhosts** is located in the home directories of the network users. It contains the remote equivalent users. See Chapter 2, "Using Network Commands."

FORMAT FOR THE /ETC/HOSTS FILE

For each machine in the network with which your node wishes to communicate, there must be an entry in the **/etc/hosts** file similar to the following example line:

```
3.Ø.Ø.16 mifa
```

where

3.Ø.Ø.16	is the internet address of the machine
mifa	is the machine's name in this example, as defined by the setuname in **/etc/rc**

For further description of the **/etc/hosts** file format, see hosts(4N) in the appropriate CTIX Operating System Manual.

NAMES AND ADDRESSES

UUCP AND NETWORK NODE NAMES

Node names are used primarily for configuring uucp connections. (See Chapter 1Ø, "Internetworking Media.") An internetwork node name is a convenience for the user to refer to a node's unique network number. The Internet Protocol, (IP) does not need the name of a node to route data. It does this using the node address. The network node name does not have to be the same as the uucp node name. Nevertheless, it is convenient to have the node's primary name the same as its uucp node name.

The name of the node is the same as the name of the machine, uname(1). Node names need not be unique throughout an internet. Each machine can have more than one name, the primary name and a number of aliases. (An alias is an alternate host name, which can be created as a convenience in addressing a host on a local network whose unique primary name is long and/or complicated.)

The node name can be a maximum of eight letters. The node name is mapped to the unique network/node address so that the user can specify either a node name or a node address to designate a unique node.

Networks can also have names for the user's convenience.

NETWORK AND NODE ADDRESSES

Even servers translate host names to addresses before doing any of their functions, using the library routine, gethostbyname; therefore the kernel never needs to know the name of its own machine or those of other hosts. Host names are strictly for network user convenience.

The system translates names into network addresses. A network address is a 32-bit value that identifies the network to which a host is attached and the logical location of the host on the net. Network addresses are entered in **/etc/hosts**, the official copy of which can be kept on a network master. This file can be distributed to each host in the local network or even to an entire internetwork. (See "Setting Up Network Database Files" below.)

The Internet Protocol uses addresses to route paths among hosts and networks. (See "Routing" below.)

If your network is to be an "in-house," freestanding network with no link to the DDN, or to an external internet, there are no special requirements for node addresses other than those conventions already existing for CTIX generally. A suggested address format is shown in Appendix G, "Internet Addresses."

An example of an internet addressing system is

    3.12

where 3 is the network number and 12 is the machine number.

A recommendation would be to use low-numbered Class A type addresses as shown in Appendix G.

If you have several LANs within your organization, it is recommended that unique network addresses be assigned even if the LANs are not currently connected, in anticipation that they may be connected eventually.

For a node that has multiple addresses for the network interfaces attached to it, it is recommended that its name be the same for all the attached of network interfaces.

Although, it is permitted for a local machine to have different names for remote machines from those used by the remote machines themselves, it is easier for the end user and the administrator if names are unique for machines commonly accessed. Otherwise a common hosts file cannot be distributed among affiliated hosts or networks.

In fact, if there is any difference at all in the node naming conventions for a local node, from the host file of the master node, a separate host file for the local node is required. However, as long as the local hosts file corresponds with the host name entries in the local route table entries, it does not matter if a remote host is known by a different name by its local users or by intermediate nodes in the routing path since the names are translated into unique addresses before sending from the local node.

DARPA Internet Addresses

If you are planning to attach to the Defense Data Network (DDN) or any other part of the DoD Internet, you should obtain a set of addresses before you add too many nodes. (See Appendix G, "Internet Addresses." For more information on the DDN see Chapter 1Ø, "Internetworking Media.")

Hosts with Multiple Network Addresses

A host must have separate addresses for each network it is on, otherwise, it could not be identified from the point of view of each separate network. In the case of a host that is on multiple networks, the host serves as a gateway to each of the networks and to itself. It has a primary address which it uses to address itself.

Mapping of Network Addresses

TCP/IP internetwork addresses require mapping to underlying networks such as Ethernet and X.25, and vice versa. On Ethernet mapping is generally performed automatically by Address Resolution Protocol (ARP). In X.25, mapping must be explicit.

ETHERNET ADDRESSES

There are three types of addresses associated with Ethernet:

- o the bus address of the media driver board, such as the Intelligent Ethernet controller. This is part of the hardware configuration described in the Release Notice for MightyFrame CTIX TCP/IP.
- o the unique Ethernet LAN address. In the MightyFrame and MegaFrame, the Ethernet board address is fixed in the board at time of manufacture. Consequently the address is not contained in the volume home block as it is in the MiniFrame.
- o the internet address, which is the combination of the network and host address. This is the address of the Ethernet network. It is used by other hosts on the network to access a particular host on a particular Ethernet network.

Ethernet Address Resolution Protocol (ARP)

The Ethernet interface drivers use ARP(7N) to dynamically map between the network address and the Ethernet address on the local Ethernet network. The protocol is not directly accessible to users.

(If your network does not support ARP, consult your technical support personnel.)

SETTING UP NETWORK DATABASE FILES

Network users use convenient host names to access a remote host and perform other network functions. The user need not know the actual numeric network address. The system maps these names to the numeric addresses which are used by the routing protocols. User programs such as ftp and telnet, also use names as a user convenience but the program itself must know the actual numeric addresses, either by being informed of it by the user or by reading **/etc/hosts.** In order to communicate, all the network hosts must know each others network and host names and addresses. Providing and distributing this information in certain system acessible files is essentially what is meant by establishing the network database files.

UPDATING THE NETWORK DATABASE FILES

The size of a network is a governing factor in determining what strategy the administrator should use in maintaining the network database files. If a network is small, made up of two or three machines, and is destined to remain this size for the indefinite future, you can use netman to set up the appropriate host databases on each machine separately. (See below, "Using Netman to Configure the Hosts Database.") If the network is large and dynamic or is expected to grow quickly, it is recommended that you maintain a central master copy of the hosts known to all nodes on the network and update each machine periodically from the master file. (See "Large Networks" below.)

LARGE NETWORKS

If the network is large or growing and changing rapidly, there should be a chief administrator responsible for maintaining a central host database. The machine on which this database is kept is referred to as the network master. (See "Network Master" below.) When a new machine is added, the local administrator retrieves the host database files from the network master. This procedure is explained below in "Retrieving host files from the Master Machine."

In a very large dynamic network, additions and changes are most likely taking place constantly. The administrators may want to update the host database files for their own network or for all the nodes in the internetwork periodically or even on a daily basis. In this case it is highly recommended that this process be automated. This step is discussed below in "Automatic Updating of Network Database Files."

Network Master

A network master is a machine in the network whose files are copied by other nodes to update their network database files. It usually has the most complete host and network files. Network master is logical concept used for the convenience of the administrators. It is not necessary to have a unique master node for the internetworking to work. In a large internet there may be several "regional" network masters depending on the system configuration and the communications patterns of the individual networks. Whether the master files are distributed locally or throughout an entire internetwork is an administrative decision.

PROCEDURAL SYNOPSIS

Set up the network database files as follows:

1. On the network master, use netman to add the new machine's name (and aliases) and address to the host file, **/etc/hosts.** (Do this even if the only machine in the network as yet is your own machine.)

   See "Using Netman to Configure the Hosts Database," below.

   If you are adding a new network, add the new network's information to the file **/etc/networks** on the network master.

2. On the new machine, use netman to add the new machine's name (and aliases) and address to the host file, **/etc/hosts** so that the host database of the local machine reflects that of the network master.

   As an alternative for larger systems, you can retrieve the **host** and **networks** files from the network master (after you have added the new machine) by using ftp. (See "Retrieving Host Files From the Network Master," below.)

3. If you have or anticipate an large dynamic system, you can automate the process. (See "Automatic Updating of the Network Database.")

USING NETMAN TO CONFIGURE THE HOSTS DATABASE

In a small network, you can use netman to add new machine names and addresses to the hosts file **/etc/hosts.** (You can also use the visual editor, vi(1), to edit the host file.)

Select the Administration function on the main menu. When the Administration menu is displayed, select one of the following as appropriate:

- o Add a new host
- o Change a host entry

Adding a New Machine to the Hosts Database

When you select "Add a new host," on the Administration menu, the form appears as shown in Figure 7-1.

```
|--| Add Host |---------------------------------|
|                                                |
|        Add a new machine  to the network       |
|                                                |
|   Host Name    |__________________|            |
|   Network Address                              |
|   Aliases                                      |
|   Comment                                      |
|                                                |
|   Use cursor keys to edit                      |
|________________________________________________|
```

Figure 7-1. Add Host Form.

1. Enter the name of the host you wish to add to the hosts database.

2. Enter the network address.

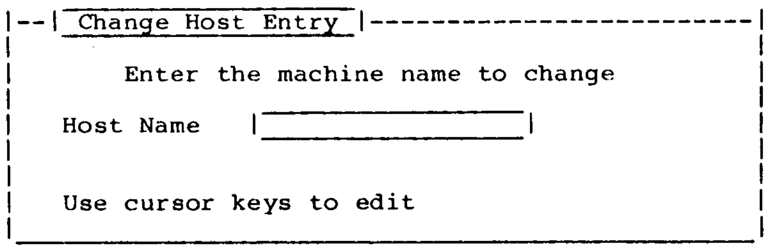
3. Enter the aliases if applicable.

4. Enter any comment.

5. Press **Next** to execute the form. (Press **Linefeed** on some terminals.) The data is entered in the **/etc/hosts** file.

Changing or Deleting a Host Entry

When you select "Change a host entry," on the Administration menu, the form appears as shown in Figure 7-2.



```
Change Host Entry

     Enter the machine name to change

  Host Name   |                  |


  Use cursor keys to edit
```

Figure 7-2. Change/Delete Host Form.

To delete the host, press **Next**. A prompt appears: "Do you wish to delete the host?" Press **Next** to execute the deletion. Press **Cancel** to leave the host entry unchanged.

To change a host entry, enter the name of the host you wish to change or delete. Press **Next** to execute the form. The data is entered in the **/etc/hosts** file.

RETRIEVING HOST FILES FROM THE NETWORK MASTER

If you have master host database, usually you will want to take advantage of it by retrieving the master database for each new machine you add to the network.

1. Before copying the network database files, have your chief administrator add your new machine to the master file **/etc/hosts.** This way when a new node is added to the network, only one file needs to be updated for the whole network to find out about the new node.

2. After you have installed your networking software and configured the local node, you can copy the hosts files from the network master so that your new machine has the information on the other machines in the network.

3. Only after starting the Ethernet board by rebooting, use _ftp_ to retrieve the host and networks files from the master machine (You must first reboot the system to enable the network command, _ftp_.)

   In this usage of _ftp_, use the internet address of the destination machine rather than its name.

   Follow the example below.

Example

(The symbol "#" is the system prompt. The commands you enter are in bold. Verbose mode is off.)

```
# ftp
ftp> open 3.Ø.Ø.16
Connected to 3.Ø.Ø.16
Name (3.Ø.Ø.16:root): root
Password (3.Ø.Ø.16:root): (enter root password for
                           machine 3.Ø.Ø.16)
ftp> cd etc
ftp> lcd /etc
ftp> get hosts
ftp> get networks
ftp> bye
#
```

AUTOMATIC UPDATING OF THE NETWORK DATABASE

It is essential to update the files containing new, changed, or deleted

- o users
- o host machines
- o uucp node names

The administrator can write a shell script that automatically updates the host file and the network file. This shell script must be installed on every system when it is configured and put in crontab(1). The database files are pulled from the master by the slave as opposed to being pushed by the master.

You can use the cron(1M) command to pick up a new edition of the master file each night.

Appendix G, "Sample /etc/gethosts File" is an example of such a shell script.

ROUTING

The network protocol translates an address into a route, which is the sequence of steps a packet must take to reach a specified address. There are two types of routes, host-specific routes and network-specific routes. A host route specifies a gateway to use from the local node to a particular host whether it be in local network or whether it is actually in another network. A network route specifies an intermediate node through which all or a portion of the traffic in the local network can be routed to on the way to destinations in another network. Each network route specifies the route to one network.

The network protocol distinguishes between these on the basis of the host routes being exceptions. When translating an address, the protocol looks in the route table first for a host route entry. If it finds one it uses it even if there is a corresponding network route.

SETTING UP ROUTING

Routing is controlled by the route(1NM) command which the administrator uses to configure the network routing tables resident at each node. The route program takes two commands: **add** a route, and **delete** a route. The command takes the form:

```
route add DEST Gateway N
```

DEST	is the name or network number of the network to which you are connecting.
Gateway	is the name or internet address of the gateway machine.

N is the number of gateways between this network and the one you are connecting to. (There must be at least one.)

You must use the metric option of the _route_ command when creating routes through gateways.

For two nodes to communicate, both must have a route to the other.

To set up routes at boot time for permanent use, add the _route_ command lines to the **/etc/rc** file after the _enpstart_ command and all the _slattach_ commands (if any).

For more information on routing, see Intro(7) in the appropriate _CTIX Operating System Manual_.

For information on _uucp_ routing, see Chapter 1Ø, "Internetworking Media.")

8 NETWORK MANAGEMENT

This chapter explains the following administrative tasks:

- o using netman to manage the node
- o setting up user and machine equivalences
- o starting up and deleting network services

This chapter also explains several techniques the administrator can use to facilitate network management duties:

- o setting up system initialization
- o automatically updating network database files
- o unloading the networking subsystem
- o setting the system prompt

This chapter also descibes some of the network services, such as and Rexecd(1NM) and some network commands, such as ifconfig(1NM).

USING NETMAN ADMINISTRATION FUNCTIONS

There are two versions of netman menus based on levels of permissions within netman, the network user and the administrator. The administrator version of netman can be accessed only with the Super User password supplied to network administrators. The various netman functions reserved for the administrator are explained in the appropriate chapter.

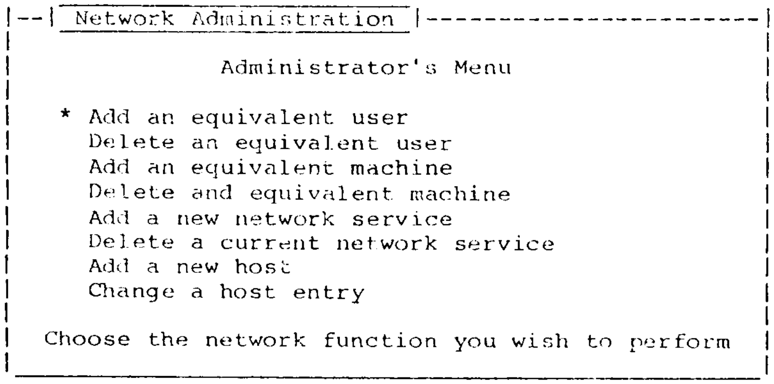
(To familiarize yourself with the network user functions of netman, see Chapter 5, "Using Netman.")

To perform administrator functions, you must know the password for the root account on your system and you must log in to the system as root before you invoke netman.

super-user can

- o equate machines
- o start and stop network services
- o modify a machine's host database
- o invoke CTIX status commands through a menu (See Chapter 9, "Network Status Monitoring.")

To call up netman's administration menu, select "Administration" on the Main menu. The Administration menu appears as shown in Figure 8-1.



```
|--| Network Administration |-------------------------|
|                                                      |
|              Administrator's Menu                    |
|                                                      |
|   * Add an equivalent user                           |
|     Delete an equivalent user                        |
|     Add an equivalent machine                        |
|     Delete and equivalent machine                    |
|     Add a new network service                        |
|     Delete a current network service                 |
|     Add a new host                                   |
|     Change a host entry                              |
|                                                      |
|  Choose the network function you wish to perform     |
|______________________________________________________|
```

Figure 8-1. Administration Menu.

You use these menu functions only on your local machine. They are not designed to perform administrative functions on remote machines even if you are the administrator of the remote machine. You can however establish a virtual terminal on the remote machine and invoke netman.

USING NETMAN ADMINISTRATION FORMS

When you select a function on the Administration Menu, the corresponding form appears. After making an entry or editing the form as appropriate, press **Next** to execute the form.

If you need to edit your entries, use the cursor keys to position over the character you wish to change.

DIRECTORY OF ADMINISTRATION MENU SELECTIONS

In Table 8-1, the selections on the administrator menu are cross-referenced to the chapter in which they are covered.

Table 8-1

Chapter Directory for Netman Administrator Menu

Administrator Function	**Chapter**
Add an equivalent user	5
Delete an equivalent user	5
Add an equivalent machine	8
Delete and equivalent machine	8
Add a new network service	8
Delete a current network service	8
Add a new host	7
Change/delete a host entry	7

MACHINE AND USER EQUIVALENCE

The first four selections on the Administration menu are for adding and deleting user and machine equivalence.

- o Add an equivalent user
- o Delete an equivalent user
- o Add an equivalent machine
- o Delete and equivalent machine

Equivalence can be established on a

- o per user basis by making an entry in the file, **.rhosts**, on the local machine (This is called user equivalence.) You can make this entry by using an editor or by using netman.

- o per machine basis by entering the name of the remote machine in the file **/etc/hosts.equiv** on the local machine

CAUTION

User and machine equivalence can be a potential weak spot in network security. If there are a lot of equivalences in a network, it is difficult to keep track of who is accessing your machine.

USER EQUIVALENCE

User equivalence is established by making an entry in the in the **.rhosts** file on the user's own machine.

(For more information and instructions on setting user equivalence, see Chapter 5, "Using Netman.")

Root Equivalence

When you are logged in as root, the Delete Equivalent User form displays the user, Root.

User Root cannot be equated by making equivalent machines. However, you can equate Roots on different machines thrugh explicit user equivalence. Also a network user can be made equivalent to Root.

Setting Up User Equivalences

You can use netman to set up user equivalences. On the Main Function Menu, select Administration. The Administration menu has two selections

- o Add an equivalent user
- o Delete and equivalent user

For instructions on setting up user equivalences. See Chapter 5, "Using Netman."

MACHINE EQUIVALENCE

Machine equivalence is an arrangement on the local machine such that whatever users are on a designated remote machine, they are to be accepted equally on the local machine, provided they have with the same name on the local machine. In order to be reciprocal, the equivalence must be set up explicitly on both the machines that want equivalences with each other. Machine equivalences can be changed at any time.

For very open systems where all machines can be equivalent, entries in a **gethosts** type command file can automate the equating of new machines. (See "Automatic Updating of the Network Database" in Chapter 7, "Network Configuration.")

Machine equivalences can be changed at any time.

SETTING UP MACHINE EQUIVALENCE

You can set up a machine equivalence invoking the Administrator menu in <u>netman</u> and selecting the appropriate function as described below.

Add Equivalent Machine Form

When you select "Add an equivalent machine," the form appears as shown in Figure 8-2.

```
|--| Add Equivalent Machine |--------------------|
|                                                |
|        Give a machine equivalence to this one  |
|                                                |
|   Machine Name     |_______________|           |
|                                                |
|                                                |
|   Use cursor keys to edit                      |
|________________________________________________|
```

Figure 8-2.   Add Equivalent Machine Form.

1. Enter the name of the remote machine you wish to make equivalent to your local machine.

2. Press **Next** to execute the form.

Delete Equivalent Machine

When you select "Delete an equivalent machine," the form appears as shown in Figure 8-3.

```
|--| Delete Equivalent Machine |-----------------|
|                                                |
|      Retract equivalence from a remote machine |
|                                                |
|   Machine Name     |_______________|           |
|                                                |
|                                                |
|   Touch "select" for choices                   |
|________________________________________________|
```

Figure 8-3.   Delete Equivalent Machine Form.

1. The form comes up already displaying the name of one of the remote machines. If this is not the name of the machine you wish to delete, enter the name of the remote machine you wish to delete as an equivalent to your local machine.

   Or press **Bound** to get the Currently Equivalent Machines display as shown in the example in Figure 8-4.

```
|--| Currently Equivalent Machines |------------|
|                                                |
|                    Machine                     |
|                                                |
|   * tom-src                                    |
|     CommDV                                     |
|     mifa                                       |
|     mitiSomm                                   |
|     mitisoft                                   |
|                                                |
|                                                |
|   Touch "select" for choices                   |
|________________________________________________|
```

Figure 8-4. Currently Equivalent Machines Display

   Using the arrow keys, move the asterisk to the machine name from which you wish to retract the privileges and press **Next**. The name is inserted into the Delete Equivalent Machine form.

2. Press **Next** to execute the deletion.

MANAGING NETWORK SERVICES

Network services are those programs which the local node is willing to provide for use by other nodes. These services can be added and deleted by the local administrator. An example is the remote login system service rlogind. Unless the local administrator adds this service, remote nodes cannot use rlogin to access the local node. All the network commands have a corresponding network service that supports the command. These services are called "demons," as is denoted by the "d" appended to the command name to form the name of the service.

Table 8-2 shows the services associated with network commands.

Table 8-2

CTIX Networking Services.

Service	Description
ftpd(1NM)	DARPA Internet File Transfer Protocol Server
olduucpd(1NM)	Network uucp server
rexecd(1NM)	remote execution server
rlogind(1NM)	remote login server
rshd(1NM)	remote shell server
rwhod(1NM)	node status server
telnetd(1NM)	telnet protocol server
tftpd(1NM)	tftp server
uucpd(1NM)	Network uucp server

USING NETMAN TO START NETWORK SERVICES

A pair of Administration menu selections (see Figure 8-1) are for provided managing network services:

- o Add a new network service
- o Delete a current network service

The most convenient way of starting and stopping network services is to is to use netman. Netman start servers immediately when you use the Add a Network Service form.

Netman also modifies the initialization file **/etc/rc** by adding a shell script. Netman also sees to it that the servers keep running if the system fails or is reset and then reinitialized.

(It is not recommended to start servers by directly editing the file **/etc/rc.** Doing so would also require rebooting to start the server.)

For a service to be started, it must be entered in the configuration file, **/etc/services.** This file does not cause the service to be started but assigns a port number to the added service. The service reads this file for information. Unless you or a programmer needs to add a new service, the file supplied with the operating system will suffice.

NOTE FOR MEGAFRAME ONLY

On the MegaFrame, do not start up demons before the intelligent boards are started. Otherwise the demons will not be aware of the existence of the intelligent boards.

Adding and Deleting a Network Service

When you select "Add a Network Service," the form appears as shown in Figure 8-5.

```
|--| Add a Network Service |--------------------------------|
|                                                           |
|             Active Network Services                       |
|                                                           |
|    * Remote Login (rlogind)                               |
|      Remote Command (rshd)                                |
|      Remote Execute (rexecd)                              |
|      Remote User Machine Status (rwhod)                   |
|      Telnet Protocol (telnetd)                            |
|      File Transfer Protocol (ftpd)                        |
|      Trivial File Transfer Protocol  (tftpd)              |
|      UUCP-Ethernet (uucpd)                                |
|                                                           |
| Move to the service you wish to cancel and touch 'next'   |
|___________________________________________________________|
```

Figure 8-5. Add Network Service Form.

The form lists all the services you can add. Netman looks in the file **/etc/services** to determine which services have been loaded on your system.

1. Select the name of the network you wish to add by pressing **Return** or the arrow keys until the asterisk (*) is positioned in front of the service you wish to add.

2. Press **Next** to execute the form. The server is started immediatedly.

If you invoke the Add Network Service form when all the services are already active, an error message is displayed.

Deleting a Network Service

When you select "Delete a Network Service," the form appears similar to the example shown in Figure 8-6.

```
|--| Remove a Network Service |---------------------------|
|                                                          |
|              Active Network Services                     |
|                                                          |
|    * Remote Login (rlogind)                              |
|      Remote Command (rshd)                               |
|      Remote Execute (rexecd)                             |
|      Remote User Machine Status (rwhod)                  |
|      Telnet Protocol (telnetd)                           |
|      File Transfer Protocol (ftpd)                       |
|      Trivial File Transfer Protocol  (tftpd)             |
|      UUCP-Ethernet (uucpd)                               |
|                                                          |
| Move to the service you wish to cancel and touch 'next'  |
|__________________________________________________________|
```

Figure 8-6.   Remove Network Service Form.

1. Select the name of the network you wish to add by pressing **Return** or the arrow keys until the asterisk (*) is positioned in front of the service you wish to delete.

2. Press **Next** to execute the form.

DESCRIPTIONS OF NETWORK SERVERS

UUCPD

(See Chapter 10, "Internetworking Media.")

REXCD

Rexecd is the server for the rexec (3N) routine. (See Chapter 13, "Using the Programmatic Interface.") The server provides remote execution facilities. Authentication is based on user names and encrypted passwords.

Rexecd listens for service requests indicated in the "exec" service specification; see services (4). When a service request is received, the rexecd initiates the protocol described in the CTIX Operating System Manual.

USING OTHER NETWORK MANAGEMENT COMMANDS

IFCONFIG(1NM)

Ifconfig is used at boot time to assign a network address to each network interface on a machine. All the network initialization commands (slattach, enpstart, and setaddr) do an implicit ifconfig to set the address and mark the interface "up." (See the appropriate CTIX Operating System Manual Note that trailers are currently ignored.)

You can use it any time later to redefine a network interface address, check the flags or modify the up/down state.

USING THE INITIALIZATION FILES TO SET UP A NODE

When the system reboots, the initialization file **/etc/initab** calls the file **/etc/drvload** to load the higher level communications protocol such as TCP/IP. It then calls **/etc/rc** (**/etc/allrc** on the MegaFrame) which can be used to set up many of the network configuration parameters.

Although almost all the steps required to set up a node can be accomplished by entering commands manually from the console, it is recommended to put as many of the basic setup commands as possible into **/etc/rc.** When you reboot the system, the commands in this file are executed. This method has the added benefit of providing for automatic reinitialization after a system failure or reset. (See Appendix C, "Sample /etc/rc File.")

Commands in **/etc/rc** can

- o identify the uname and address of the machine
- o download the media drivers onto intelligent boards
- o start network servers

For example the following commands can be put in **/etc/rc:**

- o _route_
- o _slattach_
- o _sldetach_
- o _ifconfig_

You can also set the _uucp_ node name in this file.

UNLOADING THE NETWORKING SUBSYSTEM

You can unload the TCP/IP socket networking subsystem driver only when it is not in use. The subsystem guarantees that all the hardware drivers are reset and all memory is deallocated.

SETTING THE SYSTEM PROMPT

In a large network, it is recommended that each administrator set the system prompt to the name of the node. When a user is using a virtual terminal, often the user moves the terminal from node to node. If the node name is in the prompt, the problem of remembering at any given time which node the terminal is on, is eliminated.

To set the system prompt for Bourne shell users, make the following entry in **/etc/profile**:

```
PS1 = `uname -n`$
```

The file **/etc/profile** is a global file for the Bourne shell. Therefore, this entry sets the prompt for all users of the shell.

For those using C-shell, csh(1), each user must set their prompt in their **.cshrc** file. Example entry:

```
     if (!$?HOST) then setenv HOST `uname -n`
     end if
     if (!$?HOST) then set prompt ="$HOST%"
     end if
```

Setenv is the command to set environment.

9 NETWORK STATUS MONITORING

This chapter provides information on

- o using netman for status displays
- o CTIX network status commands

OVERVIEW

Netman is introduced in Chapter 5, "Using Netman." Netman makes available in a menu-driven tabular display some of the networking status commands, such as netstat(1N), ruptime(1N), and rwho(1N). (See Table 9-1.)

Note that some status programs, such as Program Status, are run on remote machines and require equivalence for the administrator to run them.

USING NETMAN TO DISPLAY STATUS

Netman(1NM) has three status displays accessible from the Main menu shown in Figure 9-1.

```
|--| Network Manager |------------------------------|
|                                                   |
|               Main Function Menu                  |
|                                                   |
|    * Machine Status                               |
|      Network Users                                |
|      Administration                               |
|      Network Interface Statistics                 |
|                                                   |
|                                                   |
|   Choose the network function you wish to perform |
|___________________________________________________|
```

Figure 9-1. Netman Main Menu

These status display are:

- o Machine Status
- o Network Users
- o Network Interface Statistics

The first two, Machine Status and Network Users, are described in Chapter 5, "Using Netman." These can be displayed by ordinary users as well as by the administrator; however the Network Interface Statistics is primarily of interest to the administrator.

This chapter describes additional netman status displays available to the administrator. They are accessed through the Main menu selection, Network Interface Statistics. (The network user can also access this selection but the displays are designed primarily for the administrator.)

NETWORK INTERFACE STATISTICS MENU

When you select Network Interface Statistics on the Main menu, the Network Statistics menu is displayed as shown in Figure 9-2.

```
|--| Network Interface Statistics |-----------------|
|                                                    |
|    * Active Connections                            |
|      Network Interfaces                            |
|      Memory Usage                                  |
|      Routing Tables                                |
|      Protocol Statistics                           |
|                                                    |
|  Choose the network function you wish to view      |
|____________________________________________________|
```

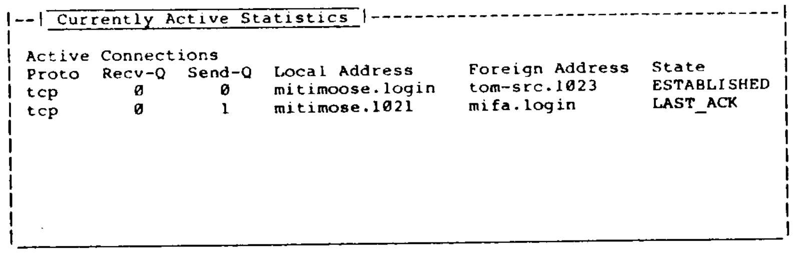
Figure 9-2. Network Interface Statistics Menu

Each of these menu selections calls up a status display. Some of these displays are also available through the netstat command. (See "Status Commands," below.)

The displays are described as follows.

✓ ACTIVE CONNECTIONS DISPLAY

When you select Active Connections on the Network Interface Statistics menu, the Active Connections display appears as in the example shown in Figure 9-3.



```
Currently Active Statistics

Active Connections
Proto  Recv-Q  Send-Q   Local Address      Foreign Address   State
tcp      0       0      mitimoose.login    tom-src.1023      ESTABLISHED
tcp      0       1      mitimose.1021      mifa.login        LAST_ACK
```

Figure 9-3. Active Connections Display

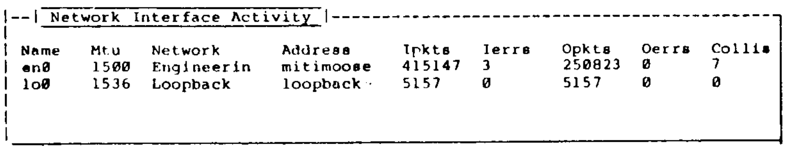
The Active Connections display is the default display of the netstat command. It displays a line of information for each active connection on the local machine under the headings described below.

Descriptions of the Display Headings

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

NETWORK INTERFACE DISPLAY

When you select Network Interfaces on the Network Interface Statistics menu, the Network Interface Activity display appears as in the example shown in Figure 9-4.



```
|--| Network Interface Activity |-----------------------------------------
|
| Name    Mtu    Network      Address      Ipkts   Ierrs   Opkts   Oerrs  Collis
| en0     1500   Engineerin   mitimoose    415147  3       250823  0      7
| lo0     1536   Loopback     loopback     5157    0       5157    0      0
|
|
|________________________________________________________________________________
```

Figure 9-4. Network Interface Activity Display

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

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

Descriptions of the Display Headings

Name	The name of the network interface. For example, en0 is the name of the first ethernet interface board.
Mtu	Maximum transmission unit (in bytes). This is the largest size permitted for any single packet sent through this interface.
Network	The name of the network address of the interface as given in **/etc/networks**.
Address	The name of the machine address of the interface as given in **/etc/networks**.
Ipkts	Input packets. The number of packets received on the interface
Ierrs	Input errors. The number of errors detected in packets of data received on this interface.
Opkts	Output packets. The number of packets transmitted on the interface
Oerrs	Output errors. The number of errors detected and corrected in packets of data transmitted on this interface.
Collis	Collisions that have occurred on the network

MEMORY USAGE DISPLAY

When you select Memory Usage on the Network Interface Statistics menu, the Memory Usage display appears as in the example shown in Figure 9-5.

```
|--| Memory Usage |-------------------------------------|
|                                                        |
|  78/320 mbufs in use:                                  |
|         34 mbufs allocated to data                     |
|         5  mbufs allocated to packet headers           |
|         13 mbufs allocated to socket structures        |
|         24 mbufs allocated to protocol control blocks  |
|         2  mbufs allocated to routing table entries    |
|   0/0 mapped pages in use                              |
|   40 Kbytes allocated to network (24% in use)          |
|   0 requests for memory denied                         |
|                                                        |
|   Touch the 'next' key to continue                     |
|________________________________________________________|
```

Figure 9-5. Memory Usage Display

The Memory Usage display shows the amount of host memory currently being used for data storage by the internetworking software. The CTIX memory is divided into structures known as mbufs. Each mbuf uses 128 bytes of memory.

Descriptions of the Display Headings

mbufs in use:	the total number of mbufs in use for the following purposes
	mbufs allocated to data
	mbufs allocated to packet headers
	mbufs allocated to socket structures
	mbufs allocated to protocol control blocks
	mbufs allocated to routing table entries

mapped pages in use

Kbytes allocated to network (n% in use)

how many mbufs are allocated to the network and how many mbufs are currently in use

requests for memory denied

how many requests for mbufs could not be fulfilled due to the lack of an available mbuf. If the network node seems to be having problems getting or losing connections, this statistic can be an important indicator.

ROUTING TABLES

When you select Routing Tables on the Network Interface Statistics menu, the Routing Tables display appears as in the example shown in Figure 9-6.

```
|--| Routing Table  |------------------------------------------------|
|                                                                    |
| Routing Tables                                                     |
| Destination        Gateway      Flags    Refcnt    Use      Interface |
| Loopback           loopback     U        0         0        lo0     |
| Engineering-Net    mitimoose    U        5         250910   en0     |
|                                                                    |
|                                                                    |
|   Touch the 'next' key to continue                                 |
|____________________________________________________________________|
```

Figure 9-6. Routing Tables Display

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

This display is also available through netstat.

Descriptions of the Display Headings

Destination	The network or machine to which this route allows you to connect.
Gateway	The name of the gateway you configured for this route. If you are directly connected, this is a local address. Otherwise it is the name of the machine through which packets must be routed.

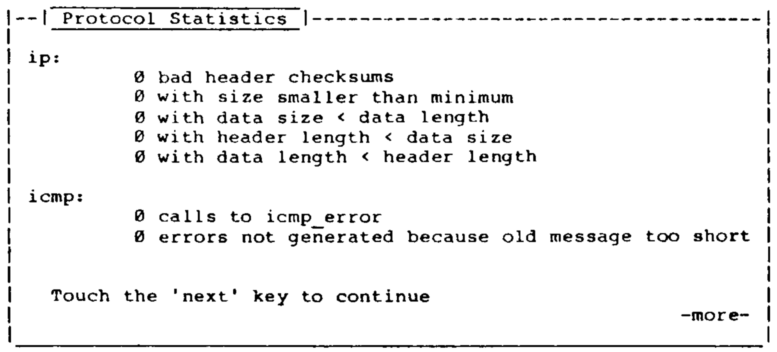
Flags	The state of the route. Valid states are: U up G the route is to a gateway N a route to a network H a route to a host
Refcnt	The current number of active connections using the route. Connection-oriented protocols normally hold on to a single route for the duration of the connection, while connectionless protocols obtain a route and then discard it as needed.
Use	The current number of packets sent using this route
Interface	The name of the physical network interface used to begin the route.

PROTOCOL STATISTICS DISPLAY

The Protocol Statistics display provides one or more display frames of protocol-specific errors. (This information is not available through netstat. The errors in the display are grouped under headings for each higher level protocol in your system. The headings are protocol-specific. Error types displayed in the example shown in Figures 9-7 through 9-9 are grouped under the following protocols:

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

When you select Protocol Statistics on the Network Interface Statistics menu, the first page of the Protocol Statistics display appears as in the example shown in Figure 9-7.



```
Protocol Statistics

 ip:
          0 bad header checksums
          0 with size smaller than minimum
          0 with data size < data length
          0 with header length < data size
          0 with data length < header length

 icmp:
          0 calls to icmp_error
          0 errors not generated because old message too short


   Touch the 'next' key to continue
                                                        -more-
```

Figure 9-7. Protocol Statistics Display, Page 1

If the prompt "-more-" is displayed in the lower righty corner, press **Next** to display the next page. Figure 9-8 and 9-8 are the second and third pages of the example display. Figures 9-8 and 9-9 are continuing pages.

```
|--| Protocol Statistics |-------------------------------------|
|                                                       -more- |
|                                                              |
|          0 errors not generated because old message was icmp |
|          0 message with bad code fields                      |
|          0 message < minimum length                          |
|          0 bad checksums                                     |
|          0 message with bad length                           |
|          0 message responses generated                       |
|                                                              |
| tcp:                                                        |
|          0 bad header checksums                              |
|          0 bad header offset fields                          |
|                                                              |
|   Touch the 'next' key to continue                           |
|                                                       -more- |
|______________________________________________________________|
```

Figure 9-8. Protocol Statistics Display, Page 2

```
|--| Protocol Statistics |-------------------------------------|
|                                                       -more- |
|                                                              |
|          0 incomplete headers                                |
|                                                              |
| udp:                                                        |
|          0 bad header checksums                              |
|          0 incomplete headers                                |
|          0 bad data length fields                            |
|                                                              |
|   Touch the 'next' key to continue                           |
|                                                              |
|______________________________________________________________|
```

Figure 9-9. Protocol Statistics Display, Page 3

Descriptions of the Display Headings

ip:	The Internet Protocol
bad header checksums	
with size smaller than minimum	This heading means there is not enough data for a header
with data size < data length	The received packet had less data than its header claimed.
with header length < data size	
with data length < header length	
icmp:	
calls to icmp_error	
errors not generated because because old message too short	

STATUS COMMANDS

The CTIX networking commands specialized to monitor network status are shown with brief descriptions and their netman equivalents are shown in Table 9-1.

Table 9-1

CTIX Networking Status Commands

Command	Netman Equivalent	Description
netstat(1N)	Network Interface Statistics	show network status (statistics)
ruptime(1N)	Machine Status	display status of nodes on local networks
rwho(1N)	Network Users	who is logged in on local network

These commands are run automatically when you select the corresponding selection from the netman menu. Netman displays the statistical data in tables. Running the operating system command directly generally gives you more options. See the machine-appropriate operating system manual for a complete description of a command and its options.

See the protocol release notice for the appropriate M/Frame machine for implementation status of these commands.

1Ø INTERNETWORKING MEDIA

This chapter gives media-specific information on the media protocol drivers and other connection services that are available for use with CTIX Internetworking.

Media currently available are are shown in Table 1Ø-1.

Table 1Ø-1

CTIX Internetworking Media Availability

	MightyFrame	**MiniFrame**	**MegaFrame**
Ethernet	X	X	X
SLIP	X	X	
X.25	X		
DDN Gateway			X
<u>uucp</u>	X	X	X

Some of the connection services shown in Table 1Ø-1 (DDN Gateway and <u>uucp</u>) are not strictly media protocols, but they are also described in this chapter because they can be used to connect nodes and networks in the CTIX Internetworking scheme.

SYSTEM PERFORMANCE AND REQUIREMENTS FOR MEDIA PROTOCOLS

Currently, all media protocols are packaged with their associated higher level protocols, TCP/IP. Therefore, see the current CTIX TCP/IP release notice for your M-Frame machine for the definitive system requirements for the media protocol.

System requirements include:

- o operating system compatibility
- o memory requirements
- o hardware requirements

For media performance specifications also see the appropriate release notice.

ETHERNET

Ethernet is a 10 megabits-per-second Local Area Network protocol that specifies the data link and physical layers of the OSI model. CTIX Ethernet subscribes to the IEEE 802.3 CSMA/CD network class. (For general information on Ethernet, see The Ethernet. For information on the CTIX Internetworking implementation of Ethernet, see Chapter 12, "System Architecture."

ETHERNET HARDWARE REQUIREMENTS

A generalized summary of Ethernet hardware requirements are given below. See the appropriate TCP/IP release notice for the definitive descriptions of hardware requirements.

All Machines

All hardware must be in place before loading the protocol drivers, media drivers, and supporting internetworking software and before reconfiguring the kernel with the internetworking software. For each Ethernet network added, one of each of the following is required:

1. IEEE 8Ø2.3-compatible transceiver

2. Transceiver drop cable that connects the machine to the transceiver

3. Ethernet backbone cable per network. (This cable connects the various transceivers.)

4. Set of transceiver tapping tools and instructions

Additional Requirements for MightyFrame

The MightyFrame TCP/IP driver can support more than one Ethernet network. Each Ethernet network requires its own hardware set. (See the Release Notice for the maximum number supported.)

1. VME Expansion consisting of an interface board and card cage. To use more than four Ethernet controllers in a system, a MightyFrame VME Cabinet is required.

2. Intelligent Ethernet controller

MiniFrame

1. MiniFrame or MiniFrame-Plus with at least one megabyte of memory.

2. Ethernet board

Additional Requirements for MegaFrame

1. Intelligent Ethernet board

2. For each Ethernet board added, an additional dedicated TP or CP is required to run a CTOS Ethernet Server. (To run the CTOS server with cluster or terminal code degrades the network performance.)

3. For memory and other requirements

4. MegaFrame expansion enclosure, if the base enclosure does not have enough slot space.

5. Ethernet on the MegaFrame requires some hardware configuration procedures for the Multibus Adapter and the Intelligent Ethernet board. These procedures are contained in the "Release Notice for MegaFrame TCP/IP."

SERIAL LINE INTERNET PROTOCOL (SLIP)

Serial Line Interconnect Protocol (SLIP) is a standard media protocol packaged with TCP/IP. Currently SLIP supports only direct (point-to-point) serial connections; however, it is inherently capable of supporting asynchronous autodial. It is available for the MightyFrame and MiniFrame. On the MightyFrame SLIP supports baud rates up to 19200 baud.

SETTING UP SLIP

To connect nodes using SLIP, you must explicitly initialize each network connection using the program slattach(1NM). Slattach sets up the appropriate interface and route descriptions within the socket driver. For each serial port that you want to use for SLIP, you must add a line to the **/etc/rc** file which runs slattach for that port.

You can also use slattach and sldetach at any time to create and remove SLIP links. Whether you choose to run these commands from **/etc/rc** or from the command line, should be determined by how often the connections change.

1. Connect the two machines with the appropriate cable attached to their communications ports. If the machines are to communicate over a leased line, connect the appropriate cables and modems.

2. Make an slattach entry in the **/etc/rc** of both machines. (This will automatically reinstate the connection upon reinitialization.) It is recommended that these lines immediately follow the line for enpstart (whether or not you are using Ethernet).

3. Reboot the system.

4. If either or both of the machines is to be a gateway for the other, establish the default route to the gateway machine through the route command. (The default route is the route that all outbound messages take that are addressed to nodes beyond the gateway, that is, all messages not intended for the local node or the gateway node.)

Using Only SLIP on the MightyFrame

If you wish to run only the SLIP media, you need to uncomment a line in the Load Driver file. The "commented out" line reads to the effect, "Load the driver if there is an Ethernet board in the system." When the system reads the uncommented line it loads the Ethernet driver automatically.

UUCP

Uucp(1C) is a batch-oriented protocol used to transfer files across phone lines using autodialers, direct lines, and networks. CTIX uucp can run on top of TCP and therefore can run over CTIX Ethernet or X.25 in a system distributed over a CTIX Internetwork. MightyFrame uucp is compatible with 4.3BSD and AT&T 3B2 releases.

NOTE

See the release notices for the machine-appropriate operating system and TCP/IP distribution for operating system compatibility and other special requirements for uucp being used with TCP/IP.

Uucp is used by electronic mail and other application systems. It can also be used directly from the shell. Once the setup described below is performed, uucp uses the internetwork communications transparently to the user processes.

UUCP ROUTING

Uucp takes the role analogous to the presentation and session layers of the OSI model. Uucp expects only "logical phone lines." Its routes can consist of multiple hops made up of both physical and logical phone lines. You must specify these routes in their entirety since uucp itself provides no routing protocol. Uucp's routing is at the presentation layer, therefore, if a lower level protocol such as TCP/IP is available for one or more hops, these protocols may do routing without uucp being aware of it.

TCP/IP can make use of Ethernet and X.25 medias and supports point-to-point links (direct cables and leased lines) through SLIP. Thus a hybrid route can be formed using uucp phone links and lines based on other connection media.

Uucp, using the data communications program uucico, places the call, logs into the remote host, negotiates with the remote host for the uucp protocol to be used in the data transfer, and begins transferring data. Some of these protocols suppress the checksum and packetizing functions because they are redundant when running over TCP.

PROCEDURE FOR SETTING UP UUCP FOR INTERNETWORKING

Uucp requires an entry for each site you wish to communicate in a the file **/usr/lib/uucp/Systems** which Uucico uses to make connections. The entry takes the form given in the procedures below.

To set up uucp to run over TCP/IP,

1. Set up uucp as directed in the appropriate release notice.

2. Use netman to start up the uucp server.

3. Add an entry for each site you wish to communicate with using uucp to **/usr/lib/uucp/Systems** according to the following specifications.

SYSTEMS FILE FORMATS

A 4.3BSD file format and an AT&T UNIX file format are provided below. Each type of line entry allows connection to both types of systems. The 4.3BSD format may be necessary in some situations because of host names. The AT&T format is provided for compatibility with an AT&T user interface.

4.3BSD Format

The 4.3BSD entry line in **/usr/lib/uucp/Systems** for each site you wish to call using uucp takes the form:

```
UcName TimetoCall DevCode Port# NetName LoginProto
```

AT&T Format

The AT&T UNIX entry line in **/usr/lib/uucp/Systems** for each site you wish to call using uucp takes the form:

```
SName TimetoCall DeviceCode "_" "_" LoginProto
```

The **Systems** file supports continuation lines.

where

SName	is the name of the site you wish to call as identified by uucp. Uucp name space antedates AF_INET name space and therefore can be different from NetName, below. This name must be unique within the file **Systems**. The actual address is taken from NetName.
TimetoCall	is the time you want to place the calls. For example, "Any"
DeviceCode	BSD systems: UCBTCP AT&T systems: TCP

"_"	unused. The double quotes ("") are place holders in the login protocol meaning "expect nothing from the remote node at this point in the exchange." There are two unused fields in the TCP style line. You can insert any string but the word "unused" is recommended.
Port#	Always enter: uucp Uucp looks up the word in /etc/services. (AT&T entries do not require a port number.)
NetName	Enter the host name from **/etc/hosts.** This entry determines the actual address of the remote host. (AT&T entries do not require a network name. The AT&T version looks up SName in **/etc/hosts.**
LoginProto	The normal uucp chat script is simplified for netowork logins. There is no need to wait for any return strings, although login and password prompts are sent.

Entry Example

```
rochester Any TCP uucp ur-seneca "-" uucplogin "-" uucppasswd
```

In this example, rochester is a name known to uucp but known to the Internet as ur-seneca. Rochester also happens to be the name of a different Internet host.

The double quotes ("") are place holders in the login protocol meaning "expect nothing from the remote node at this point in the exchange." At the end of the login protocol, the local node is logged into the remote node.

(For more information on uucp, see uucp and related entries in the appropriate CTIX Operating System Manual and CTIX Administrator's Reference Manual.)

HANDLING SITES WITH EARLIER RELEASE LEVELS

Some sites you wish to communicate with may not have updated their release version of uucp. This may be indicated by the fact that the above systems file entry does not work in their case. You can still communicate with them is you make entries in the following earlier format:

```
UucpName TimetoCall DeviceCode BaudRate RemoteLoginAcct
```

where

UucpName	is the same as the current format above
TimetoCall	is the same as the current format above
DeviceCode	Enter: INET
BaudRate	Always enter: 96ØØ
RemoteLoginAcct	Enter: nuucp

If you need to communicate with an "old site" for which you have made an entry in the above format, you must start olduucpd(1NM). Use netman. See Chapter 8, "Network Management."

UUCPD

Uucpd is the network server for CTIX file transfer using the uucp user interface and protocols. It is similar to rshd(1NM) server, except:

1. The remote socket need not be privileged.

2. The shell invoked must be **/usr/lib/uucp/uucico.**

A network uucp connection is indicated with the INET keyword in **/usr/lib/uucp/L.sys.** Uucpd is normally executed by the startup file, **/etc/rc.**

See also the other "uu" entries in the CTIX Operating System Manual.

CTIX TCP/IP-X.25 INTERFACE

The MightyFrame CTIX TCP/IP-X.25 Interface is an optional internetworking media product which enables CTIX TCP/IP to access the MightyFrame CTIX X.25 Network Gateway. TCP/IP thereby makes use of X.25 as a media protocol to extend CTIX Internetworking to Wide Area Networks (WANs). Thus CTIX hosts and Local Area Networks (LANS) that run network servers and application systems on top of TCP/IP can communicate through X.25 packet-switched Public Data Networks (PDNs). Using X.25 as a connection media can be cost effective in joining more than two LANs into a larger internetwork. See Figure 10-1.

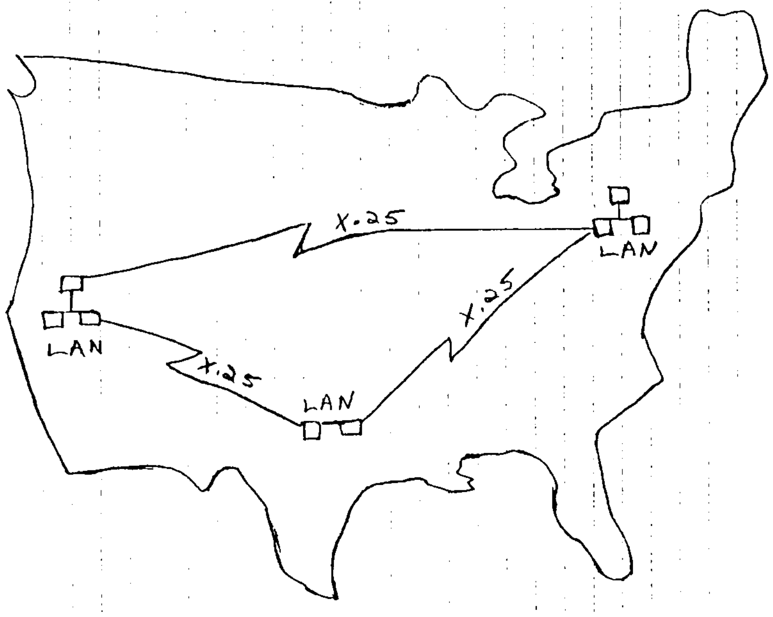


Figure 10-1. Wide Area X.25 Used to Connect LANs.

When you establish the appropriate routes between hosts across the gateway(s), the TCP/IP-X.25 Interface becomes fully transparent to the network user and to user application systems.

X.25 GATEWAY NODES

The CTIX TCP/IP-X.25 Interface runs only on a MightyFrame. There is no comparable product for the MegaFrame and MiniFrame. Therefore only a MightyFrame can act as a host for TCP/IP using X.25 as an internetworking media protocol. (See Figure 6-2.)

The CTIX TCP/IP-X.25 Interface can coexist on a host with other protocols, such as SNA and the SLIP and Ethernet media protocols.

To connect a TCP/IP networked MightyFrame to the PDN, the MightyFrame X.25 Network Gateway is required in addition to the CTIX TCP/IP-X.25 Interface. The MightyFrame CTIX X.25 Network Gateway is a loadable driver that supports X.25 applications by providing access to X.25 PDNs.

POINT-TO-POINT X.25 NETWORKING

The MightyFrame and the MegaFrame have different software configurations for their X.25 gateway capabilities. The MightyFrame version is called the MightyFrame CTIX X.25 Network Gateway. The MegaFrame CTIX X.25 Interface interfaces to the CTOS X.25 Network Gateway to provide the same functionality at the MightyFrame counterpart. A product similar to the MightyFrame X.25 Network Gateway is also available for the MiniFrame.

The MightyFrame and MegaFrame X.25 gateways are compatible in a point-to-point X.25 network configurations. Both X.25 gateways can be used for point-to-point X.25 connections to other hosts or to an X.25 PDN without any further interfaces. (See Figure 10-2.)

Figure 10-3 shows a MightyFrame connected point-to-point to a PDN.

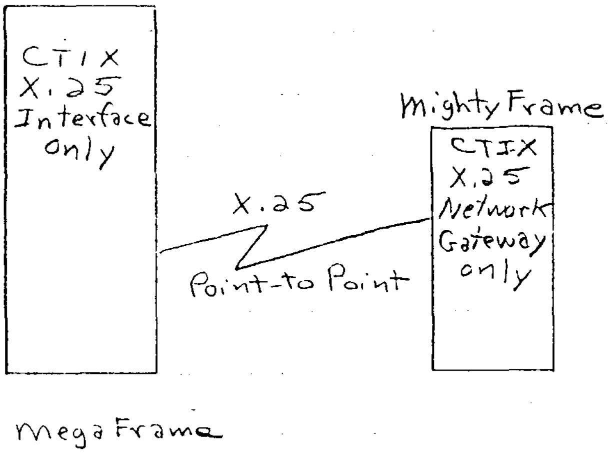


Figure 10-2. CTIX Systems Linked by Point-To-Point X.25.

A point-to-point X.25 network is a separate and distinct network without routing capability other that that provided by the X.25 PDN. CTIX Internetworking can be implemented only using TCP/IP.

A point-to-point X.25 network and a TCP/IP internetwork can share the same MightyFrame X.25 Network Gateway. See the example shown in Figure 10-3. TCP/IP can share a MightyFrame CTIX X.25 Network Gateway on the same host with other service applications such as the CTIX X.3/X.28/X.29 PAD.

The network user cannot directly access standard PDN facilities such as the CTIX X.3/X.28/X.29 PAD using standard TCP/IP commands. However, it is possible to use TCP/IP virtual terminal capabilities to log onto a host that is directly connected to a PDN and has the PAD available for terminals.

INTERNETWORKING WITH ETHERNET AND X.25

MegaFrame and MiniFrame hosts on Ethernets connected to the MightyFrame gateway hosts can be internetworked across an X.25 PDN. An example of such a configuration is shown in Figure 10-3. All features available on Ethernet can be accessed transparently across the PDN using a MightyFrame as a gateway.

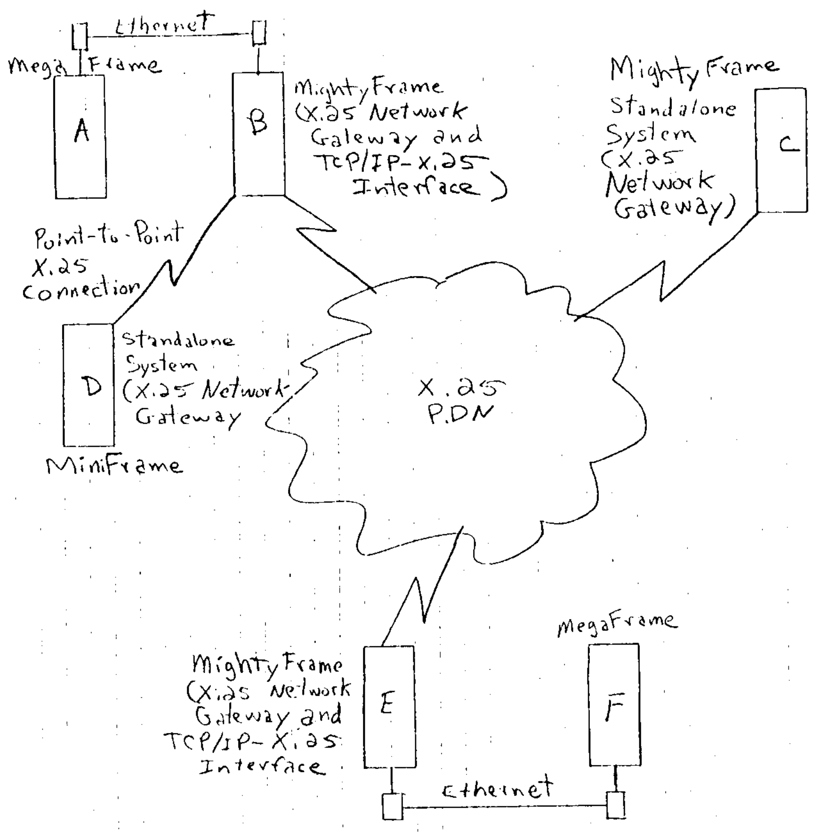

Ethernet
MegaFrame
A
B
MightyFrame (X.25 Network Gateway and TCP/IP-X.25 Interface)
MightyFrame
Standalone System (X.25 Network Gateway)
C
Point-to-Point X.25 Connection
D
Standalone System (X.25 Network Gateway
MiniFrame
X.25 P.DN
MegaFrame
MightyFrame (X.25 Network Gateway and TCP/IP-X.25 Interface
E
F
ethernet

Figure 10-3. Example X.25/Ethernet Internetworking Configuration

Figure 10-3 shows an internetwork configuration example consisting of

- o a TCP/IP Internetwork using Ethernet and X.25 as media protocols (hosts A, B, E. and F)

- o two separate non-TCP/IP X-25 point-to-point networks consisting of a standalone MightyFrame (host C) connected to a PDN and a standalone MiniFrame (host D) connected point-to-point to a MightyFrame (host B)

In the TCP/IP internetwork, note that the MegaFrame host A can access host F, another MegaFrame on an Ethernet, across the X.25 PDN.

Note also that point-to-point connections can coexist with a TCP/IP internetwork. Both the TCP/IP internetwork and the link between host B and host D share the MightyFrame X.25 Network gateway in host B. However, the non TCP/IP host D cannot route through B to access the PDN.

The non-TCP/IP host C, can access X.25 application systems on host B through the PDN but cannot access host D through the MightyFrame X.25 Network Gateway installed on host B.

ECONOMY AND ERROR HANDLING FEATURES

The MightyFrame CTIX TCP/IP-X.25 Interface automatically closes an active virtual circuit that remains inactive for five minutes.

The MightyFrame CTIX TCP/IP-X.25 Interface recovers from transmission errors encountered in an active X.25 virtual ciruit by automatically closing the faulty circuit and opening a new one. This process is entirely transparent to the user program.

RELATED X.25 DOCUMENTS

For more information and installation instructions for the MightyFrame CTIX TCP/IP-X.25 Interface, see the "Release Notice for The MightyFrame CTIX TCP/IP-X.25 Interface."

Additional information on the X.25 media protocol, is contained in Chapter 12, "System Architecture."

For more information and installation instructions for the MightyFrame X.25 Network Gateway, the MegaFrame CTIX X.25 Interface, and the CTOS X.25 Network Gateway, see the appropriate manual and release notice, as listed in "Related Documents" at the front of this manual.

DDN NETWORK GATEWAY

The DDN Network Gateway is an optional, separately packaged product that links a standalone MegaFrame to the Defense Data Network. It is available only on the MegaFrame. Other hosts on a CTIX internetwork cannot access the DDN across a MegaFrame on the internetwork since MegaFrame does not support CTIX IP gateways. (See Chapter 12, "System Architecture.")

The Defense Data Network (DDN) is a set of communications capabilities which links together computer systems within the Department of Defense (DoD). The DDN allows users of these computer systems to send mail and exchange files between systems and to access other computers on the Internet in interactive virtual terminal sessions. The DDN is based on TCP/IP and X.25.

(For more information, see Chapter 12, "System Architecture." and DDN MegaFrame Reference Manual.)

11 INTERNETWORKING CONCEPTS

This chapter reviews some basic network concepts relating to CTIX internetworking, TCP/IP, and network gateways. At the generalized level of discussion in this chapter, CTIX internetworking closely parallels corresponding concepts in the DARPA Internet Program and 4.3BSD UNIX. More CTIX-implementation-specific information on CTIX protocol configurations is given in Chapter 12, "System Architecture."

OVERVIEW

TCP (Transmission Control Protocol) and IP (Internet Protocol) form an end-to-end transmission and routing protocol that supports CTIX commands and applications. TCP/IP was originally developed for the DoD for use in ARPANET, the first major network to use packet-switching technology. (ARPA stands for Advanced Research Projects Agency.) (Packet switching is an alternative to the circuit switching technology used in telephone and telex systems in which a dedicated communication path is allocated to communications between two users for the duration of a communication.) Packets are also called datagrams. A message can be broken up into several packets, or datagrams.

The Defense Data Network (DDN) is based on ARPANET standard TCP/IP. The DDN is the implementation stage in the evolution of work done by ARPA of the DoD on packet switching network technology. (SRI, under contract with the DoD, publishes a standard for TCP/IP.)

Currently, TCP/IP are also widely used in UNIX environments and are included in UC Berkeley UNIX. CTIX TCP/IP is similar to and usually compatible with other systems running TCP/IP over Ethernet.

THE OSI MODEL AND CTIX INTERNETWORKING

Although CTIX Internetworking is not strictly an implementation of the well known seven-layer OSI network model, it is helpful to compare them in explaining CTIX Internetworking concepts. Figure 11-1 shows the OSI model and the roughly corresponding CTIX Internetworking layers.

OSI Model	Layer	CTIX Internetworking
Application	7	Applications, e.g.: Applications <u>ftp</u>, <u>telnet</u> (Applications in CTIX include level 6 and some functions of 5.)
Presentation	6	
Session	5	Socket Interface and Transport Control Protocol
Transport	4	
Network	3	Internet Protocol
Link	2	Media Protocol
Physical	1	Physical Media

Figure 11-1. Comparison of OSI Model and CTIX Internetworking Model

Layer four in CTIX Internetworking can be implemented in different protocols. In this chapter, TCP is used as an example of layer four and is explained in some detail. The CTIX Internet Protocol (IP) which implements layer three is also explained. TCP/IP makes possible the end-to-end nature of CTIX Internetworking.

The data link layer are implemented in CTIX Internetworking as Media protocols, such as Ethernet. They are often implemented in hardware. Media Protocols are explained in chapter. 1Ø "Internetworking Media."

There are also several higher level specialized protocols for specific applications such as terminal traffic (telnet) and file transfer (ftp) and protocols for other network functions such as gateway status monitoring and control and error reporting, but, in this manual, these are not usually referred to as protocols, but rather as programs or services. The same applies to the link level protocols, such as Ethernet, which are referred to as a media protocol or simply a media.

HOW PROTOCOLS COMMUNICATE

Protocols communicate logically only with their counterpart protocols and physically only with the layers directly above and below themselves. For example, TCP, at layer 4, the Transport layer of host A, communicates logically only with its counterpart in the remote host, but, to do so, it communicates physically with the layer directly beneath it, the IP at the network layer 3. IP in turn, IP passes the message from TCP down to its neighboring layer, and so on. The physical layer transmits the message to host B. (See Figure 11-2).

The message passes up through the layers of host B to arrive at the TCP for host B which is able to decode the message and disposition it appropriately. The intervening layers are unaware of the contents of the message. They only decode its destination address to the extent they must to pass it to the next layer.

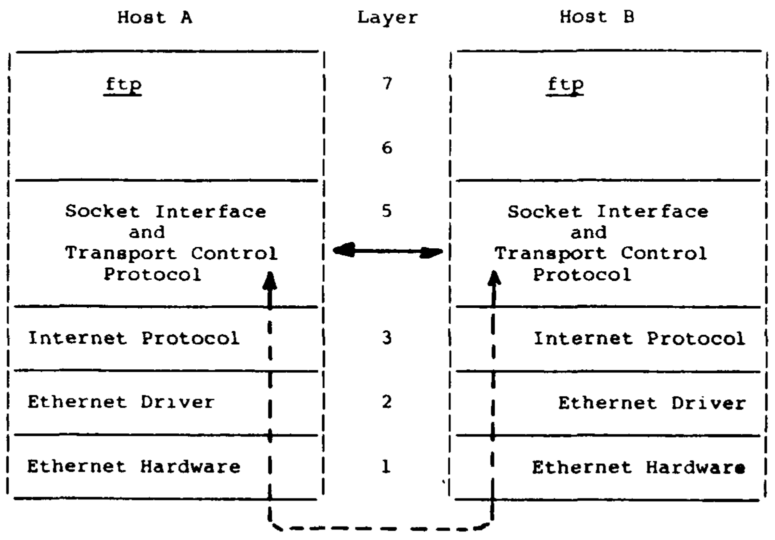


Figure 11-2. Example of How Protocols Communicate

Each layer therefore must understand the language of the adjoining upper and lower layers in addition to performing its own proper functions.

GATEWAYS

While Figure 11-1 does not describe the topology or composition of an internetwork using TCP and IP, such an internetwork can be composed of different CTIX-compatible networks running different local protocols but tied together by a higher level end-to-end protocol such as TCP/IP. To unify a potentially diverse networks into a single internetwork, network gateways are used. A gateway is a switching node that connects two or more networks, especially if they use different protocols. (If a gateway runs on a dedicated processor, the processor is also considered a component of the gateway.)

GATEWAY ROUTING PROTOCOLS

The networks connected by gateways can employ the same protocols or different protocols. If the same, the gateway must contain modules of the protocol being used. For example, if the networks both use TCP/IP, the gateway must contain TCP and IP modules. If different, the gateway must contain services capable of converting from one network protocol to another.

When a message arrives at a gateway, the gateway determines the destination network and encapsulates the internet protocol with the protocol header appropriate to the new network.

IP GATEWAY PROTOCOL

The network level protocol, IP, is designed to act as a gateway. All CTIX Internetworking nodes, including gateways, contain an internet module. This IP module contains the appropriate software services for switching and retransmitting packets (or virtual circuits) to their next gateway, to the next network, or to the destination host. Gateways are placed wherever necessary to implement the desired topology and configuration of an internet. A gateway is usually one of the network hosts but it can reside in its own dedicated processor.

TCP AND IP PROTOCOLS

TCP and IP are actually separate protocols that work together implementing the major lower level functions in CTIX internetworking.

TRANSMISSION CONTROL PROTOCOL (TCP)

Transmission Control Protocol is a transport level, connection-oriented protocol that provides highly reliable end-to-end message transmission between hosts in packet-switched networks and in interconnected systems, or internets. TCP interfaces on one side with user or applications processes and on the other side to a lower level protocol such as Internet Protocol (IP). TCP communicates asynchronously with the application process and the IP or other network level protocols. TCP is totally byte-stream oriented.

TCP operates a sufficiently high level (OSI level 4 and 5) to be media independent. Lower layers can support hard-wired (direct), circuit-switched, and packet-switched communications links. Thus an internetwork using TCP to provide an end-to-end transport service, can be made up of multiple subnets using a variety of media protocols.

A TCP module resides at each node in the internet that communicates with other TCP nodes. TCP can reside in the host or in a front-end communications processor.

To provide its service under the expected circumstances of the datagram model, TCP implements mechanisms for the following functions:

- support for interprocess communication (IPC) and connection functions
- basic data transfer and transmission
- end-to-end reliability
- multiplexing, flow control, and message sequencing
- precedence and security
- out of bound data

Interface with Application Process

TCP provides the basis for CTIX interprocess communications over the internet. The interface between TCP and application processes consists of a set of calls much like the calls an operating system provides to a process or manipulating files. For example, there are calls to open and close connections and to send and receive data on established connections. (A connection is a logical concept rather than virtual circuit as in X.25.

Datagrams are routed individually and dynamically over the best available routing path. They are reassembled at the receiving end.)

TCP Ports and the Socket Interface

TCP provides a set of numbered ports to identify and be used by the calling processes. A TCP port is not a hardware communications port such as an RS-232-C port. A TCP port is the portion of a socket that specifies which logical input or output channel of a process is associated with the data.

A socket is an address which specifically includes a port identifier, that is, the concatenation of an internet address with a TCP port. Port connections are displayed in the Active Connections Display of netman. See Chapter 9, "Network Status Monitoring." (See also "Glossary.") A process, for example, could be a number of terminals talking to a host.

For more information on sockets and how TCP uses them, see Chapter 13, Using the Programmatic Interface."

TCP and Reliable Transmission

The primary purpose of TCP is to provide a reliable, secure, virtual circuit connection service between pairs of communicating processes. (Security provisions such as limiting user access to certain nodes can be implemented at the TCP layer. See Chapter 13, Using the Programmatic Interface.")

TCP is concerned only with total end-to-end reliability. It makes few assumptions about the possibility of obtaining reliable datagram service from the lower protocols. If a datagram is sent across an internet to a remote node, the intervening networks do not guarantee delivery. Likewise, the sender of the datagram has no way of knowing the routing path used to send the datagram. Source to destination reliability is provided by TCP.

Reliability is achieved through checksums (error detection codes), positive acknowledgment of data received, and retransmission of unacknowledged data.

Flow Control

Flow control is accomplished by allowing the receiver to regulate the data rate. TCP provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission.

For more information on TCP, see Internet Protocol Transition Workbook, "Transmission Control Protocol," RFC-793.

INTERNET PROTOCOL

Internet Protocol is the network level protocol designed for packet-switched networks. The IP limits itself to only the delivery of datagrams through an internet. Reliability is the responsibility of TCP) As with TCP, there must be an Internet Protocol module at each node and each gateway that communicates using Internet Protocol. This module is called the internet module.

The IP addresses, routes, and forwards datagrams to the next gateway or destination host via the local network interface. Internet gateway functions are performed at the IP layer.

IP functions are:

- o addressing
- o security classification, and compartmentation of TCP segments
- o internet datagram routing
- o communications with gateways and host protocol modules
- o fragmentation and reassembly

Addressing

An IP message retains the TCP fixed-length, 45-bit address across the internet. The address consists of a network number and a local host address (32 bits) and a 16-bit port address. The local address field contains the address of the host within the network. There are three classes, or formats, of addresses to accommodate networks of different size. (See Appendix G, "Network Addressing.")

TCP and User Datagram Protocol (UDP) also use a 16-bit number to address a connection.

The internet module maps internet addresses to local net addresses. Local nets and gateways map from local net addresses to routes. A host can have several physical interfaces to different networks in the internetwork with each interface having its own logical internet address.

Hosts Having Multiple Addresses

In the case of a host that is on multiple networks, the host serves as a gateway to each of the networks and to itself. It has a primary address which it uses to address itself.

If the packet is for itself (a local process), the host sends the packet to the loopback driver of the routing code, which chains the output back to the input.

Routing

An internet module can be located in a gateway or a network host. It decides the routing path for the datagram, packages it with the next address, and forwards the resulting internet datagram to the next gateway or to the destination host. IP treats each datagram as an independent entity during all phases of routing. (TCP sees the datagram only at the endpoints.) Internet modules reassemble datagrams into the original messages only at the destination host.

An internet module packages received datagrams with the appropriate internet header, containing the address of the final destination. Such a datagram is called and internet datagram.

At the destination host, the local network interface strips the internet datagram of its local net header and hands it to the internet module.

The internet module determines whether the datagram is for an application system in the local host. The internet module passes the message (after reassembling it if necessary) to the application system in response to a system call. If the datagram is not for the local host, the IP passes it on.

For more information on Internet Protocol, see RFC 791, "Internet Protototocol," in the Internet Protocol Transition Workbook.

Fragmentation

In a packet-switched network, packets can be sent by different routes and are retransmitted if necessary. If datagrams arrive at the destination host out of sequence, the IP reassembles them into the original message for the destination host. Some networks have different message sizes. If necessary at the destination host, the IP fragments incoming datagrams to the size required by the destination network.

12 SYSTEM ARCHITECTURE

This chapter describes some machine-specific implementations of the CTIX internetworking protocols.

OVERVIEW

Functionally CTIX internetworking software consists of several layers from top to bottom:

- o user command interface
- o network servers
- o system calls, and library subroutines
- o sockets interface
- o transport service protocol
- o internet protocol
- o physical media

As a consequence of significant architectural differences among the CTIX machines, the implementation of these layers varies considerably from machine to machine. Also the configuration of the internetworking software for a given CTIX machine can vary from release to release and is dependent on the current version of the operating system for the respective machines. Therefore it is essential to consult the current release notices for your system's internetworking software and operating system.

The major aspects of different implementations of the CTIX internetworking software currently used are described in this chapter.

SOFTWARE CONFIGURATION AND FUNCTIONAL IMPLEMENTATION

It is important to distinguish between the software configuration of the networking software and its functional implementation in the architecture of the particular CTIX machine. Software configuration means how the various protocols are bundled or unbundled in the loadable protocol drivers. For example, on the MightyFrame and MiniFrame machines, the TCP/IP kernel driver, the Ethernet driver are bundled in the same loadable/unloadable driver. (The CTIX machine-specific version of the driver is different for each machine type.)

Software configurations are best known by referring to the appropriate release notice for the networking software.

Functional implementation means how the functional components of the internetworking software relate to the machine architecture and to other functional components of the internetworking software. Some examples are as follows.

- In the MiniFrame, all the networking protocols run in the kernel. But the MightyFrame release also contains the executive image for the MightyFrame's intelligent Ethernet controller card and the program that downloads the Ethernet driver onto the intelligent controller during system startup.

- Each machine type has its own limitations as to the number of TCP virtual circuits it can support.

- All the CTIX machines have their own versions of the Ethernet processor board.

Figures 12-1 through 12-3 show CTIX Internetworking functional implementations on the different CTIX machines.

MINIFRAME

In common with the other CTIX machines, user programs, such as the network servers, run above the surface of the kernel. The socket layer interfaces between the user processes and the transport and session protocols such as TCP and UDP. The socket driver functions at a level analogous to the OSI session layer.

In the MightyFrame and MiniFrame, the kernel driver that runs the TCP/IP protocol, the Ethernet and SLIP network interfaces are all in one loadable/unloadable socket driver.

In the MiniFrame, TCP and UDP, the Internet Protocol and the Media protocols, Ethernet and SLIP all run in the kernel. See Figure 12-1.

MIGHTYFRAME

In the MightyFrame, the transport and network protocols run in the kernel. The SLIP and X.25 media protocols also run in the kernel. The Ethernet driver is downloaded to run on an intelligent controller. MightyFrame can support multiple Ethernet controllers. (See Figure 12-2.)

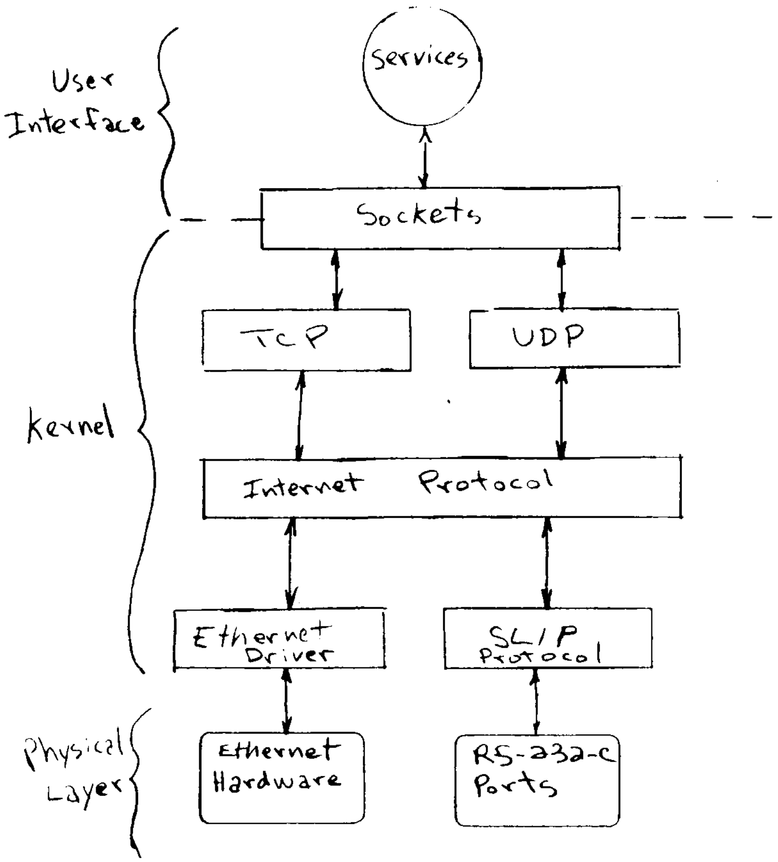


Figure 12-1. MiniFrame Internetworking Implementation.

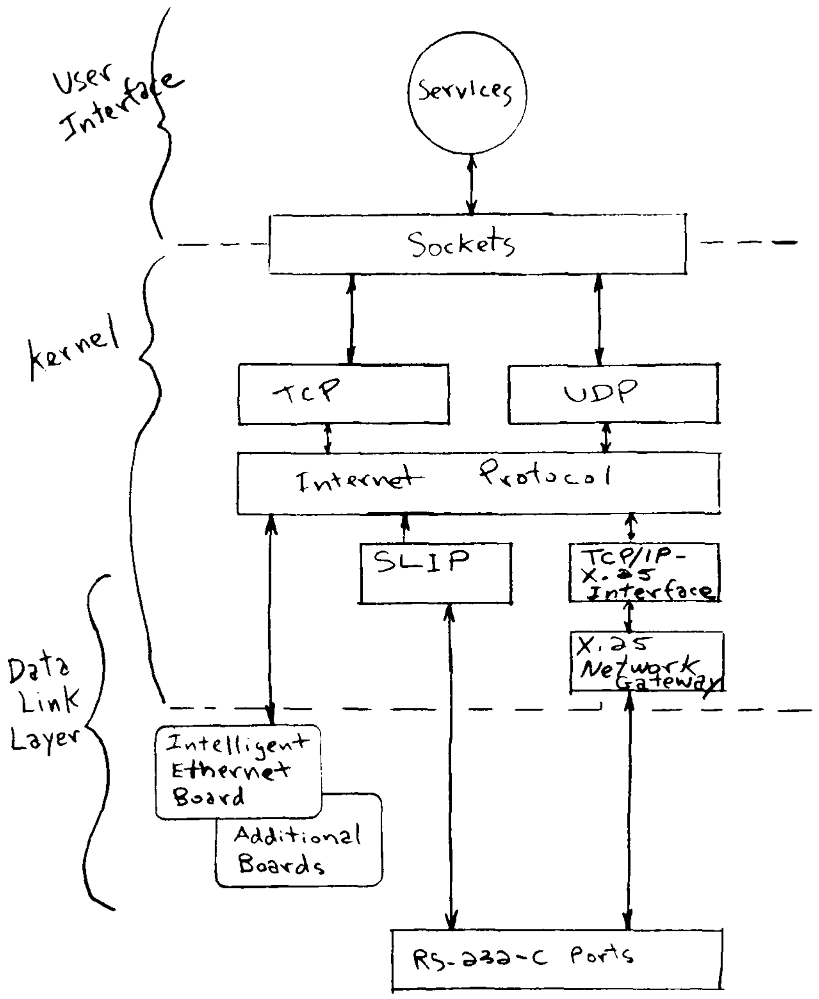


Figure 12-2. MightyFrame Internetworking Implementation.

MIGHTYFRAME CTIX IP GATEWAYS

Only the MightyFrame architecture is fully capable of providing a CTIX internetworking gateway. An example of a CTIX IP gateway module linking an X.25 network and an Ethernet network is shown in Figure 12-3.

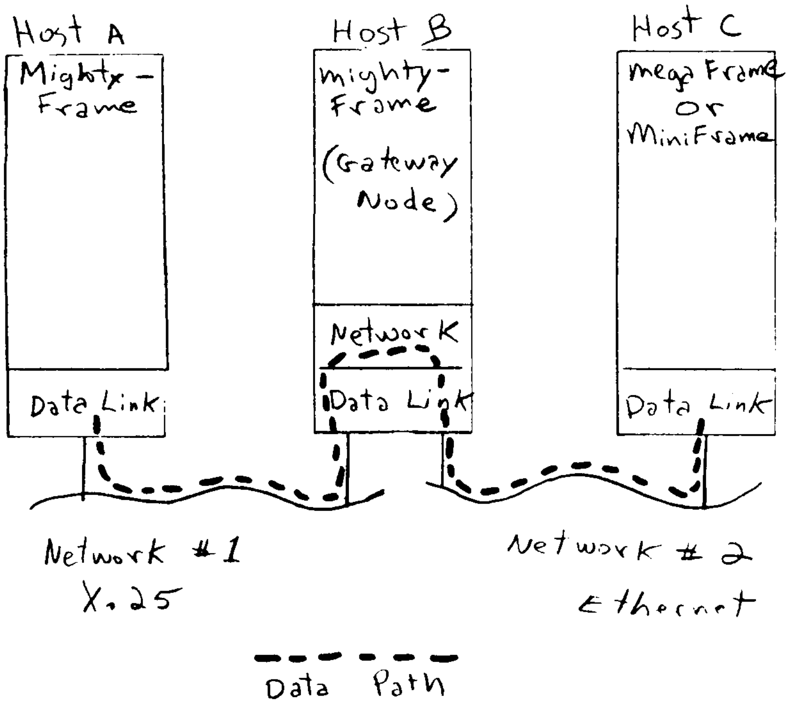


Figure 12-3. Routing Path in An Internetwork Using the Internet Protocol as a Gateway

The data at the data link layer from the MightyFrame (Host A) on the X.25 network is routed by the network layer of the MightyFrame (Host B) to the data link layer of the MegaFrame (Host C) at the data link layer. (The dashed line describes the data path.)

MEGAFRAME

In MegaFrame the socket drivers are part of the TCP/IP loadable driver. After loading, the socket drivers are relinked to the kernel resulting in a new kernel configuration.

Currently MegaFrame TCP/IP does not support User Datagram Protocol (UDP) and Internet Protocol (IP). Consequently, those programs and functions that rely on these protocols (such as ruptime, rwho, and calls using datagram sockets) are not implemented. See the release notice for your release version of the product to determine which protocols are supported.

MegaFrame Ethernet is currently implemented in a single subsystem with TCP/IP. Both run on the Intelligent Ethernet Card which is decoupled from the CTIX kernel. See Figure 12-4.

MEGAFRAME TCP/IP AND ETHERNET

The TCP/IP and Ethernet driver is downloaded to run on an intelligent controller board. The board plugs into a Multibus adapter. The Multibus adapter attaches to the MegaBus of the MegaFrame. (See Figure 12-5.)

The Multibus adapter is tightly coupled with a dedicated Terminal Processor (TP) or Cluster Processor (CP). A dedicated TP or CP runs the CTOS Ethernet Server which enables the sharing of the same Ethernet board by multiple Applications Processors (APs). For each Ethernet board added, an additional dedicated TP or CP is required to run a CTOS Ethernet Server.

Note that to run the CTOS server with cluster or terminal code degrades the network performance.

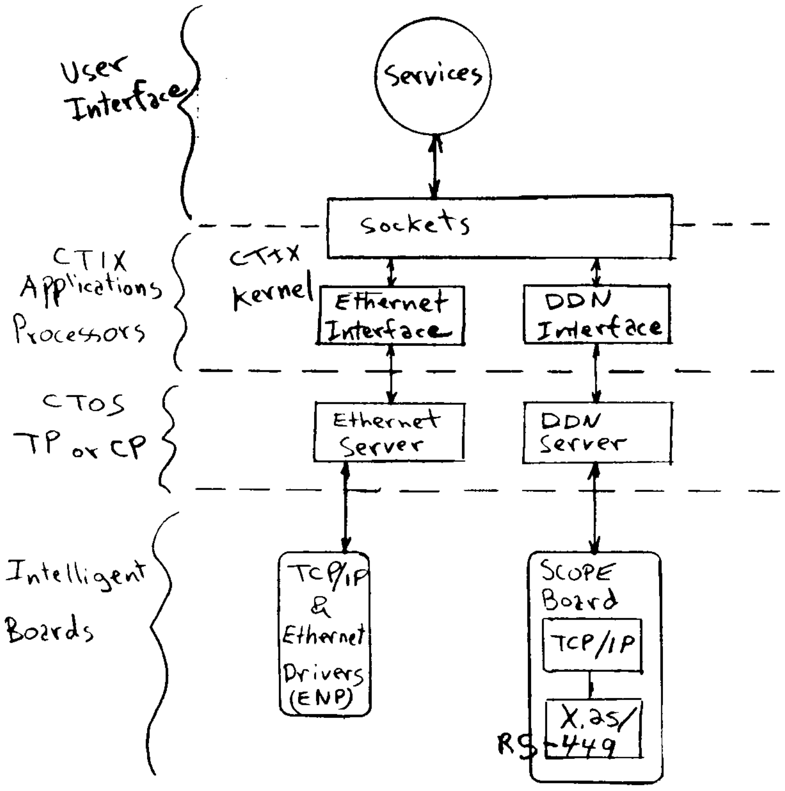


Figure 12-4. MegaFrame Internetworking Implementation.

```
+-------------+    +-------------+    +-------------+
|    A P      |    |    A P      |    |    A P      |
|             |    |             |    |             |
|  C T I X    |    |  C T I X    |    |  C T I X    |
|             |    |             |    |             |
+-------------+    +-------------+    +-------------+
     |  |               |  |                |  |
     |  |               |  |                |  |
<----+  +---------------+  +----------------+  +------->
             M     E     G     A     B     U     S
<-------------+  +----------------+  +----------------->
              |  |                |  |
              |  |                |  |
         +-------------+     +-------------+
         |             |     |             |
         |  C T O S    |     |  MULTIBUS   |
         |             |     |  ADAPTER    |--+
         |  TP OR CP   |     |             |  |
         +-------------+     +-------------+  |
                                |  ETHERNET   |
                                |    BOARD    |
                                +-------------+
                                      | |
                                      | |
                   ---------------------------------------
                   ---------------------------------------
                                 E T H E R N E T
```

Figure 12-5. MegaFrame Ethernet Configuration Example.

MEGAFRAME A NON-GATEWAY MACHINE

Its architecture does not permit the MegaFrame to be used as a gateway machine. Its IP module, which runs on an intelligent controller board, is not available in the kernel for routing as is the MightyFrame's; therefore, any such routing scheme using a MegaFrame would have to be implemented at the session level using sockets and is currently not supported.

13 USING THE PROGRAMMATIC INTERFACE

This section describes the programmatic interface supported by CTIX Internetworking. The programmatic interface provides a framework for interprocess communications both within the same host and across CTIX Internetworking protocols.

OVERVIEW

A client process, such as a user application system, usually needs to communicate with a server process to perform its functions. One way that this interprocess communication (IPC) is provided for in CTIX is pipes, already discussed in Chapter 2, "Using Network Commands." CTIX Internetworking also provides a more flexible and powerful independent subsystem especially designed to support IPC in a distributed environment. This subsystem is called the sockets interface, or "sockets." The CTIX Internetworking sockets interface comprises the programmatic interface and the basis for IPC both within a host and across an internet.

The CTIX Internetworking sockets interface is an implementation compatible with the Berkeley 4.3BSD socket mechanisms distributed for the Internet Domain. The Berkeley sockets interface was designed to interface TCP/IP and other protocols with the UNIX kernel. The CTIX Internetworking version is also capable of supporting other communications protocols.

The CTIX internetworking software distribution contains the socket subsystem protocol that is necessary to support CTIX interprocess communication, networking system calls and the library subroutines.

The TCP/IP loadable driver implements the socket abstraction and protocol. By linking their programs with the facilities in the library, **libsocket**.a, programmers can write their own distributed programs using interprocess communication. The library routines call the kernel directly. The calling programs can be written in the C language or another language.

The library, **libsocket**, contains system calls and library routines. The calls are linked to the actual system call primitives in the kernel. The system calls perform basic functions for an application system.

The library routines are commonly used name handling routines. The library routines are listed and described below.

(For information on the internal system implementation of the sockets interface, see 4.2BSD Networking Implementation Notes.)

SOCKETS

A socket is a software entity that provides the basic building block for interprocess communications. Sockets allow processes to rendezvous in a CTIX name space through which they exchange data. A socket is an endpoint of communication between processes. For Internet addresses, a fully named pair of sockets uniquely identify a connection between two communicating sides:

```
<<node.port> <node.port>>
```

where node is the four-byte network address and port is two bytes identifying the network interface. The socket on the left is the local socket and the socket on the right is the remote, or foreign, socket.

The activity status of the user processes can be seen in the Active Connections display of netstat. If both sides of a socket pair are operating on the local machine, each is listed separately. (See Chapter 9, "Network Status Monitoring."

Sockets exist within communications domains. A communications domain is system of communications properties of the communicating processes and of the underlying communications facilities of the domain itself. One such property is the scheme used to name sockets. Currently CTIX supports only sockets existing in the name space of the Internet Domain. Sockets normally exchange data only with sockets in the same domain; otherwise a translation process is required.

TYPES OF SOCKETS

A socket has a type and one or more associated processes. Sockets are typed by the communications properties visible to the programmer. Usually a socket type is associated with the particular protocol which the socket supports. Processes usually communicate between sockets of the same type. Three types of sockets are available to the programmer:

- stream socket
- datagram socket
- raw socket

Stream Sockets

A (SOCK_STREAM) is the recommended and most commonly used type. In the AF_INET communications domain, a stream socket takes advantage of the inherent reliability of the transport level byte stream protocol, TCP. It provides bidirectional, sequenced, and unduplicated flow of data without boundaries.

Datagram Sockets

A datagram socket (SOCK_DGRAM) supports bidirectional flow of data in the datagram model of the network level protocol. (See Chapter 11, "Internetworking Concepts.") Record boundaries are preserved. The receiving process must perform resequencing, elimination of duplicates, and reliability assurance. The datagram socket can be used in applications where reliability of an individual packet is not essential, for example, in broadcasting messages for the purpose of updating a status table.

Unreliable datagram protocol (UDP) supports the datagram socket. (See "Connectionless Sockets," below.)

Raw Sockets

With a raw socket (SOCK_RAW), the programmer has access to the underlying communications protocols which support sockets, such as the IP. Raw sockets can be implemented variously depending on the interface provided by the communications protocols chosen.

Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more esoteric facilities of an existing protocol.

Other Types of Sockets

Reliable datagram and sequenced packet sockets are not currently available in CTIX.

HOW SOCKETS ARE CONTROLLED

The socket subsystem keeps track of sockets through a set of protocol control blocks (PCBs) which describe the processes and their addresses. When a process opens a socket, the subsystem checks its chain of active PCBs to see if the socket pair is already being used.

SYSTEM CALLS

System calls are used to perform interprocess communications primarily by manipulating sockets. The linker editor, ld(1), searches these functions under the -l socket(2N) option. The calls directly invoke CTIX system primitives in the kernel.

Together with the other CTIX system calls, the CTIX networking system calls are documented in Chapter 2 of the appropriate CTIX Operating System Manual. In that manual, all networking system calls are designated 2N. For example:

accept(2N)

The networking system calls are listed and briefly described in Table 13-1. In the following paragraphs the calls are discussed under the various process operations.

Table 13-1.

CTIX Networking System Calls

Call	**Description**
accept(2N)	accept a connection on a socket
bind(2N)	bind a name to a socket
connect(2N)	initiate a connection on a socket
getpeername(2N)	get name of connected peer
getsockname(2N)	get socket name
getsockopt(2N), setsockopt	get and set options on sockets
listen(2N)	listen for connections on a socket
recv, recvfrom(2N)	receive a message from a socket
send, sendto(2N)	send a message to a socket
shutdown(2N)	shut down part of a full-duplex connection
socket(2N)	create an endpoint for communication

ERROR RETURNS

All the system calls return -1 in case of error. The error number is put in the variable errno. Error numbers are defined in **<sys/errno.h>**.

In a model exchange between a calling process and a serving process, the client is the active process, the server is the passive process. The client and the server use different types of socket calls that are appropriate to their roles. Some of the system calls that can be used are paired and arranged in logical order as follows:

Serving Process	Client Process
create socket	create socket
bind	bind
listen	connect
accept	
read	write
write	read
close	close

CREATING A SOCKET

To create a socket, use the socket system call:

```
s = socket(domain, type, protocol);
```

Where

domain	In CTIX, the domain is always AF_INET (address family_Internet domain). The manifest constants are named AF_whatever because they indicate the "address family" to use in interpreting names.
type	Types are o SOCK_STREAM o SOCK_DGRAM o SOCK_RAW

type If the protocol is unspecified (a value of Ø), the system selects an appropriate protocol from those available to support the requested socket type. The system returns a small integer descriptor, or handle, to use in later system calls which operate on sockets. This is equivalent to a file descriptor. See open(2).

To select a particular protocol, select from those defined in **sys/in.h**. You can also use one of the library routines described below, such as getprotobyname.

Example: `s = socket(AF_INET, SOCK_STREAM,Ø);`

Selecting a Protocol

To obtain a particular protocol one selects the protocol number, as defined within the communication domain. For the Internet domain, the available protocols are defined in **sys/in.h** or, better yet, one may use one of the library routines discussed below, such as getprotobyname (getprotoent(3N)):

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
 ...
pp = getprotobyname("tcp");
s = socket(AF_INET, SOCK_STREAM, pp->p_proto);
```

Socket Creation Errors

ENOBUFS	The system lacks sufficient memory for an internal data structure
EPROTONOSUPPORT	Unknown protocol or protocol not supported.
EPROTOTYPE	Socket type request has no supporting protocol.
ENODEV	The socket driver is not loaded.

BINDING SOCKET NAMES

A socket is created without a name, but, to be used, it must be given a name. The bind call is used to assign a name to a socket on the local side of a connection:

```
bind(s, name, namelen);
```

s	S is the socket descriptor. (See "Creating a Socket," above.)
name	The bound name is a variable length byte string to be interpreted by the supporting protocol(s) according to the domain type. In the Internet domain, names contain an Internet address and port number.
namelen	Namelen is the length of the name.

You do not have to specify an address unless you want a certain one. To bind an Internet address, the call is:

```
#include <sys/types.h>
#include <sys/in.h>
...
struct sockaddr_in sin;
...
bind(s, &sin, sizeof (sin));
```

To determine what to place in the address _sin_, see "Networking Library Routines," below. If you set _sin_ to Ø, the system binds to the server for you and returns the address it used.

GETTING A CONNECTION

Once a process has bound a local socket, the process can rendezvous with an unrelated foreign process. Usually the rendezvous takes the form of a client server relationship.

The Client

The client completes the other side of the socket pair when it requests services from the server by initiating a connection by issuing a _connect_ call:

```
struct sockaddr_in server;
connect(s, &server, sizeof (server));
```

If the client process's socket is unbound when it issues the _connect_ call, the system automatically selects a name and binds the socket. If the socket is successfully associated with the server, data transfer can begin. If not, an error is returned. Only the active process uses _connect_.

The Server

A completed connection is identified by a unique pair of sockets, each socket being an endpoint associated with one of the reciprocating processes. For the server to receive a client's connection, the server must issue two system calls after binding its socket. The first is to indicate a willingness to listen for incoming connection requests; the second is to accept the client's connect. To "listen" is to passively wait to accept a connection from a client process:

```
listen(s, 5);
```

The second parameter, 5, indicates the maximum number of outstanding connections which can be queued awaiting the acceptance of the server. This limit prevents processes from hogging system resources. Should a connection be requested while the queue is full, the server does not refuse the connection, but ignores the messages which comprise the request. This forces the client to retry the connection request and gives the server time to make room in its queue.

Had the connection been returned with the ECONNREFUSED error, the client would be unable to tell if the server was up or not. As it is now it is still possible to get the ETIMEDOUT error back, though this is unlikely. The backlog figure supplied with the listen call is limited by the system to a maximum of five pending connections on any one queue. This avoids the problem of processes hogging system resources by setting an infinite backlog, then ignoring all connection requests.

Wildcard Addressing a Socket Location. A server can underspecify its location to service incoming service requests from multiple network interfaces by using the wild card symbol (*). A service such as ftp can installed only once on a host which is connected to multiple network interfaces Ftp, can listen on all the network interfaces:

```
<<*.21> <*.*>>
```

This tuple signifies that the local ftp on port 21 is listening on multiple interface addresses for whatever client processes that wish to connect.

To name a socket that listens on all network interfaces, the Internet address INADDR_ANY must be bound. If a listening port is not specified, the system assigns one. (Wildcarding is discussed further in "IPC Programming Techniques," below.)

Accepting a Connections. With the socket marked as listening, the server can now accept a connection:

```
fromlen = sizeof (from);
snew = accept(s, &from, &fromlen);
```

The server returns a new descriptor to the client on receipt of a connection (along with a new socket). If the server wishes to find out who its client is, it may supply a buffer for the client socket's name. Fromlen is a value-result parameter initialized by the server to indicate how much space is associated with from (the client). It is modified on return to reflect the true size of the name. Only a passive process uses accept.

Accepts normally blocks. That is, the call to accept will not return until a connection is available or the system call is interrupted by a signal to the process. Further, there is no way for a process to indicate it will accept connections from only a specific individual, or individuals. It is up to the user process to consider who the connection is from and close down the connection if it does not wish to speak to the process. If the server process wants to accept connections on more than one socket, or not block on the accept call, there are alternatives which are considered in "IPC Programming Techniques," below.)

Servers often bind multiple sockets. When a server accepts a connection, it usually spins off (forks) a process which is the connected socket. The parent then goes back to listening on the same local socket.

Connection Errors

Of the many errors that can be returned when a connection fails, the following are the most common.

ETIMEDOUT	After failing to establish a connection during a period of time, the system decided there was no point in retrying any more. The cause for this error is usually that the remote host is down or that problems in the network resulted in transmissions being lost.
ECONNREFUSED	The host refused service for some reason. This error is usually caused by a server process not being present at the requested host.

ENETDOWN or EHOSTDOWN

Status information received by the client host from the underlying communication services indicates the net or the remote host is down.

ENETUNREACH or EHOSTUNREACH

These operational errors can occur either because the network or host is unknown (no route to the host or network is present) or because status information to that effect has been delivered to the client host by the underlying communication services.

TRANSFERRING DATA

When a connection is established, data flow can begin using a number of possible calls. If the peer entity at each end of a connection is anchored (that is, there is a connection), a user can send or receive a message, without specifying the peer, by using the read and write:

```
write(s, buf, sizeof (buf));
 read(s, buf, sizeof (buf));
```

The calls send and recv are virtually identical to read and write, except that a flags argument is added.

```
send(s, buf, sizeof (buf), flags);
recv(s, buf, sizeof (buf), flags);
```

One or more of the flags can be specified as a nonzero value as follows.

SOF_OOB	Send/receive out of band data. Out of band data is specific to stream sockets.

SOF_PREVIEW	Look at data without reading. When specified in a recv call, any data present is returned to the user but treated as though still "unread." The next read or recv call applied to the socket will return the data previously previewed.
SOF_DONTROUTE	Send data without routing packets. (Used only by the routing table management process.)

DISCARDING SOCKETS

If a socket is no longer of use, the process can discard it by applying a close to the descriptor:

```
close(s);
```

If data is associated with a socket which promises reliable delivery (a stream socket), the system will continue to attempt to transfer the data. However, after a fairly long period of time, if the data is still undelivered, it will be discarded. If a user process wishes to abort any pending data, it can apply a shutdown on the socket prior to closing it. Shutdown causes any data queued to be immediately discarded. The call format is:

```
shutdown(s, how);
```

where how is:

0 if the user no longer wishes to read data

1 if no more data will be sent

2 if no data is to be sent or received.

CONNECTIONLESS SOCKETS (SOCK_DGRAM)

UDP Datagram sockets provide only connectionless interactions. When using datagram sockets, the programmer does not have to issue a connect call before sending. A datagram socket provides a symmetric interface to data exchange. Datagram processes are still likely to be client and server, there is no requirment for connection establishment. Each message includes the destination address.

Sending from Datagram Sockets

Datagram sockets are created and name bound exactly as are stream sockets but to send data from a datagram socket, the process uses the sendto primitive:

```
    sendto(s, buf, buflen, flags, &to, tolen);
```

The parameters are the same as those described for send, above, except the to and tolen values are used to indicate the intended recipient of the message.

When using an unreliable datagram interface, it is unlikely any errors will be reported to the sender. If information at the sending node indicates that the message cannot be delivered, for instance, when a network is unreachable, the call returns - 1 and the global value errno will contain an error number.

Receiving on Datagram Sockets

To receive data on an unconnected datagram socket, use recvfrom:

```
 recvfrom(s, buf, buflen, flags, &from, &fromlen);
```

The parameters are as described above. Note that fromlen is handled in the value-result manner described under "Connections," above.

Using Connect on a Datagram Socket

Datagram sockets can also use connect to associate a socket with a specific address. In this case, any data sent on the socket is automatically addressed to the connected peer, and only data received from that peer will be delivered to the user.

Only one connected address is permitted for each socket (that is, no multicasting). Connect requests return immediately; the system merely records the peer's address, as compared to a stream socket where a connect request initiates establishment of an end to end connection. Other of the less important details of datagram sockets are described in "IPC Programming Techniques," below.

If connect is used with a datagram socket, read and write and send and recv can be used to transfer data.

INPUT/OUTPUT MULTIPLEXING

(The select call is not currently supported.)

An application system can multiplex I/O requests among multiple sockets and/or files by using select:

```
select(nfds, &readfds, &writefds, &execptfds,
&timeout);
```

Select takes three bit-masks as arguments, one for each of the following:

- o the set of file descriptors for which the caller wishes to be able to read data on
- o those descriptors to which data is to be written
- o to indicate which exceptional conditions are pending

Bit masks are created by or-ing bits of the form "1 << fd". That is, a descriptor fd is selected if a 1 is present in the fd'th bit of the mask. The parameter nfds specifies the range of file descriptors (that is, one plus the value of the largest descriptor) specified in a mask.

A timeout value may be specified if the selection is not to last more than a predetermined period of time. If timeout is set to Ø, the selection takes the form of a poll, returning immediately. If the last parameter is a null pointer, the selection will block indefinitely. (A return takes place only when a descriptor is selectable, or when a signal is received by the caller, interrupting the system call.) Select normally returns the number of file descriptors selected. If the select call returns due to the timeout expiring, then a value of -1 is returned along with the error number EINTR.

Select provides a synchronous multiplexing scheme. Asynchronous notifications of output completion, input availability, and exceptional conditions is possible through use of the SIGIO and SIGURG signals. (See "Using Special Signals, below.")

NETWORKING LIBRARY ROUTINES

The definitive descriptions of the CTIX networking library routines are contained in chapter 3 of the appropriate CTIX Operating System Manual, together with the other CTIX library routines. In those manuals, all networking library routines can be distinguished by the designation 3N. For example:

gethostent(3N)

OVERVIEW

Most of the routines in **libsocket** are concerned with providing a uniform interface between the application system and the network database. The routines perform commonly used file name handling and manipulation. The primary uses of these routines are to locate and construct network addresses.

These routines have been designed with flexibility in mind. As more communication protocols become available, the same user interface will be maintained in accessing network-related address databases. The only difference is the values returned to the user. Since these values are normally supplied by the system, users should not need to be directly aware of the communication protocol and/or naming conventions in use.

Locating a service on a remote host requires many levels of mapping before client and server may communicate. A service is assigned a name which is intended for human consumption; for example, "the login server on host CT". This name, and the name of the peer host, must then be translated into network addresses which are not necessarily suitable for human consumption.

Finally, the address must then used in locating a physical location and route to the service. The specifics of these three mappings is likely to vary between network architectures. The mapping process can be further complicated by additional layers required to interface disparate systems and for security considerations as described in the following example.

For instance, it is desirable for a network to not require hosts be named in such a way that their physical location is known by the client host. Instead, underlying services in the network may discover the actual location of the host at the time a client host wishes to communicate. This ability to have hosts named in a location independent manner may induce overhead in connection establishment, as a discovery process must take place, but allows a host to be physically mobile without requiring it to notify its clientele of its current location.

The CTIX networking library routines are C programming language function calls, which you can call from your program but which do not go into the kernel to be executed. They are relinked with the library at link-time. The interface can support a variety of protocols. The file **<netdb.H>** must be included when using any of these routines.

Standard routines are provided for mapping:

- o host names to network addresses
- o network names to network numbers
- o protocol names to protocol numbers,
- o service names to port numbers and the appropriate protocol to be used with the server

The categories of library routines are listed with brief descriptions in Table 13-2. Each of these categories contains multiple functions.

Table 13-2.

C Programming Language Function Calls

byteorder(3N)	convert values between host and network byte order
gethostent(3N)	get network host entry
gethostname(3N)	get name of current host
getnetent(3N)	get network entry
getprotoent(3N)	get protocol entry
getservent(3N)	get service entry
inet(3N)	Internet address manipulation
rexec(3N)	return stream to a remote command

(For more information on these routines see the appropriate CTIX Operating System Manual.)

MAPPING HOST NAMES

A host name to address mapping is represented by the hostent data structure:

```
struct     hostent {
           char          *h_name;
           char          **h_aliases;
           int           h_addrtype;
           int           h_length;
           char          *h_addr;
};
```

where

*h_name	is the official name of the host.
**h_aliases	is the alias list.
h_addrtype	is the host address type.
h_length	is the length of address.
*h_addr	is the address.

The routine gethostbyname(3N) takes a host name and returns a hostent structure. Gethostbyaddr(3N) maps host addresses into a hostent structure. If a host has more than one address having the same name, gethostbyname returns the first entry in /etc/hosts. If this is not adequate, the lower level routine gethostent can be used. An example of a routine to obtain a hostent structure for a host on a particular network is given in Figure. 13-1.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/in.h>
#include <netdb.h
...
struct hostent *
gethostbynameandnet(name,net)
        char *name;
        int net;
{
        register struct hostent *hp;
        register char **cp;

        sethostent(0);
        while ((hp = gethostent()) != NULL) {
                if (hp->h_addrtype != AF_INET)
                        continue;
                if (strcmp(name, hp->h_name)) {
                        for (cp = hp->h_aliases; cp && *cp != NULL; cp++)
                                if (strcmp(name, *cp) == 0)
                                goto found;
                        continue;
                }
        found:
                if (inet_netof(*(struct in_addr *)hp->h_addr)) == net)
                        break;
        }
        endhostent(0);
        return (hp);
}
```

(Inet netof(3N) is a standard routine which returns the network portion of an Internet address. (See inet(3N)).

Figure 13-1. Example of a Hostent Routine

MAPPING NETWORK NAMES

As for host names, routines for mapping network names to numbers, and back, are provided. These routines return a netent structure:

The network number is limited to 32 bits.

```
struct      netent {
            char         *n_name;
            char         **n_aliases;
            int          n_addrtype;
            int          n_net;
};
```

where

*n_name	is the official name of net.
**n_aliases	is the alias list.
n_addrtype	is the net address type.
n_net	is the network number.

The routines getnetbyname(3N), getnetbynumber(3N) and getnetent(3N) are the network counterparts to the host routines described above.

MAPPING PROTOCOL NAMES

For protocols the protoent structure defines the protocol-name mapping used with the routines getnetent(3N) and getprotoent(3N):

```
struct      protoent {
            char         *p_name;
            char         **p_aliases;
            int          p_proto;
};
```

where

*p_name	is the official protocol name.
**p_aliases	is the alias list
p_proto	is the protocol number.

MAPPING SERVICE NAMES

Information regarding services is a bit more complicated. A service is expected to reside at a specific "port" and employ a particular communication protocol. This view is consistent with the Internet domain, but inconsistent with other network architectures. Further, a service may reside on multiple ports or support multiple protocols. If either of these occurs, the higher level library routines will have to be bypassed in favor of homegrown routines similar in spirit to the "gethostbynameandnet" routine described in Figure 13-1, above. A service mapping is described by the servent structure:

```
struct      servent {
            char         *s_name;
            char         **s_aliases;
            int          s_port;
            char         *s_proto;
};
```

where

*s_name	is the official service name
**s_aliases	is the alias list.
s_port	is the port number.
*s_proto	is the protocol to use

The routine _getservbyname_(3N) maps service names to a servent structure by specifying a service name and, optionally, a qualifying protocol. Thus the call:

```
     sp = getservbyname("telnet", (char *)0);
```

returns the service specification for a telnet server using any protocol, while the call

```
       sp = getservbyname("telnet", "tcp");
```

returns only that _telnet_ server which uses the TCP protocol.

The routines _getservbyport_(3N) and _getservent_(3N) are also provided. The _getservbyport_ routine has an interface similar to that provided by _getservbyname_; an optional protocol name may be specified to qualify lookups.

HANDLING NETWORK DEPENDENCIES

With the support routines described above, an application program should rarely have to deal directly with addresses. This allows services to be developed as much as possible in a network independent fashion. It is clear, however, that purging all network dependencies is very difficult. So long as the user is required to supply network addresses when naming services and sockets, there will always some network dependency in a program. For example, the normal code included in client programs, such as the remote login program, is of the form shown in Figure 13-2.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <netdb.h>
 ...
main(argc, argv)
       char *argv[];
{
       struct sockaddr_in sin;
       struct servent *sp;
       struct hostent *hp;
       int s;
       ...
       sp = getservbyname("login", "tcp");
       if (sp == NULL) {
              fprintf(stderr, "rlogin: tcp/login: unknown service\n");
              exit(1);
       }
       hp = gethostbyname(argv[1]);
       if (hp == NULL) {
              fprintf(stderr, "rlogin: %s: unknown host\n", argv[1]);
              exit(2);
       }
       bzero((char *)&sin, sizeof (sin));
       bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
       sin.sin_family = hp->h_addrtype;
       sin.sin_port = sp->s_port;
       s = socket(AF_INET, SOCK_STREAM, 0);
       if (s < 0) {
              perror("rlogin: socket");
              exit(3);
       }
       ...
       if (connect(s, (char *)&sin, sizeof (sin)) < 0) {
              perror("rlogin: connect");
              exit(5);
       }
       ...
}
```

Figure 13-2. Remote Login Client Code.

If we wanted to make the remote login program independent of the Internet protocols and addressing scheme, within the limitations of the current IPC organization, we would be forced to add a layer of routines which masked the network dependent aspects from the mainstream login code.

Manipulating Byte Strings and Handling Byte Swapping

Aside from the address-related data base routines, there are several other routines available in the run-time library which are intended mostly to simplify manipulation of names and addresses. Table 13-3 summarizes the routines for manipulating variable length byte strings and handling byte swapping of network addresses and values.

The byte swapping routines are provided because the operating system expects addresses to be supplied in network order. On a VAX, or machine with similar architecture, this is usually reversed. Consequently, programs are sometimes required to byte swap quantities. The library routines which return network addresses provide them in network order so that they may simply be copied into the structures provided to the system. This implies users should encounter the byte swapping problem only when interpreting network addresses. For example, if an Internet port is to be printed out the following code would be required:

```
  printf("port number %d\n", ntohs(sp->s_port));
```

On some machines other than the VAX and Intel microprocessor-based machines, these routines are defined as null macros.

Table 13-3.

C Run-Time Routines.

Call	Synopsis
memcmp(s1, s2, n)	compare byte-strings; Ø if same, not Ø if otherwise: returns a number less than Ø if S1 < S2; returns a number greater than Ø if S1 > S2.
memcpy(s1, s2, c, n)	copy n bytes from s1 to s2
memset(s, c, n)	zero-fill n bytes starting at base
htonl(val)*	convert 32-bit quantity from host to network byte order
htons(val)*	convert 16-bit quantity from host to network byte order
ntohl(val)*	convert 32-bit quantity from network to host byte order
ntohs(val)*	convert 16-bit quantity from network to host byte order

* These calls are no-ops on 68ØØØ machines. They are operations in Intel microprocessor-based architectures. They should be included for portability.

USING THE CLIENT/SERVER MODEL

The most commonly used paradigm in constructing distributed applications is the client/server model. In this scheme client applications request services from a server process. This implies an asymmetry in establishing communication between the client and server. In the following paragraphs we will look more closely at the interactions between client and server, and consider some of the problems in developing client and server applications.

OVERVIEW

Client and server require a well known set of conventions before service may be rendered (and accepted). This set of conventions comprises a protocol which must be implemented at both ends of a connection. Depending on the situation the protocol may be symmetric or asymmetric.

In a symmetric protocol, either side may play the master or slave roles. In an asymmetric protocol, one side is immutably recognized as the master, with the other the slave. An example of a symmetric protocol is the telnet protocol used in the internet for remote terminal emulation. An example of an asymmetric protocol is the internet file transfer protocol, ftp. No matter whether the specific protocol used in obtaining a service is symmetric or asymmetric, when accessing a service there is a "client process" and a "server process". We will first consider the properties of server processes, then client processes.

SERVER PROCESS

A server process normally listens at a well known address for service requests. ("Well known" means that port assignments for services are usually stable and can be seen in **/etc/services.** See "Typical TCP/IP Exchange Between Machines," below.) Alternative schemes which use a service server can be used to eliminate a number of server processes clogging the system while remaining dormant most of the time. For example, the Berkeley implemention of the Xerox Courier protocol uses the latter scheme. (This is an example only; CTIX does not currently support Courier.)

When using Courier, a Courier client process contacts a Courier server at the remote host and identifies the service it requires. The Courier server process then creates the appropriate server process based on a database and "splices" the client and server together, voiding its part in the transaction. This scheme is attractive in that the Courier server process may provide a single contact point for all services, as well as carrying out the initial steps in authentication.

However, while this is an attractive possibility for standardizing access to services, it does introduce a certain amount of overhead due to the intermediate process involved. Implementations which provide this type of service within the system can minimize the cost of client server rendezvous. The portal notion described in the 4.2BSD System Manual embodies many of the ideas found in Courier, with the rendezvous mechanism implemented internal to the system.

In 4.3BSD, most servers are accessed at well known Internet addresses or UNIX domain names. When a server is started at boot time, it advertises it services by listening at a well know location. For example, the remote login server's main loop is of the form shown in Figure 13-3.

```
main(argc, argv)
	int argc;
	char **argv;
{
	int f;
	struct sockaddr_in from;
	struct servent *sp;

	sp = getservbyname("login", "tcp");
	if (sp == NULL) {
		fprintf(stderr, "rlogind: tcp/login: unknown service\n");
		exit(1);
	}
	...
#ifndef DEBUG
	<<disassociate server from controlling terminal>>
#endif
	...
	sin.sin_port = sp->s_port;
	...
	f = socket(AF_INET, SOCK_STREAM, 0);
	...
	if (bind(f, (caddr_t)&sin, sizeof (sin)) < 0) {
		...
	}
	...
	listen(f, 5);
	for (;;) {
		int g, len = sizeof (from);

		g = accept(f, &from, &len);
		if (g < 0) {
			if (errno != EINTR)
				perror("rlogind: accept");
			continue;
		}
		if (fork() == 0) {
			close(f);
			doit(g, &from);
		}
		close(g);
	}
}
```

Figure 13-3. Remote Login Server.

Once a server has established a pristine environment, it creates a socket and begins accepting service requests. The bind call is required to insure the server listens at its expected location. The main body of the loop is fairly simple:

```
for (;;) {
        int g, len = sizeof (from);

        g = accept(f, &from, &len);
        if (g < 0) {
                if (errno != EINTR)
                        perror("rlogind: accept");
                continue;
        }
        if (fork() == 0) {
                close(f);
                doit(g, &from);
        }
        close(g);
}
```

An accept call blocks the server until a client requests service. This call could return a failure status if the call is interrupted by a signal such as SIGCLD (to be discussed in "Programming Techniques," below.). Therefore, the return value from accept is checked to insure a connection has actually been established. With a connection in hand, the server then forks a child process and invokes the main body of the remote login protocol processing. Note how the socket used by the parent for queueing connection requests is closed in the child, while the socket created as a result of the accept is closed in the parent. The address of the client is also handed the doit routine because it requires it in authenticating clients.

CLIENT PROCESS

The client side of the remote login service is shown in Figure 13-2. One can see the separate, asymmetric roles of the client and server clearly in the code. The server is a passive entity, listening for client connections, while the client process is an active entity, initiating a connection when invoked.

To consider more closely the steps taken by the client remote login process, as in the server process, the first step is to locate the service definition for a remote login:

```
sp = getservbyname("login", "tcp");
if (sp == NULL) {
        fprintf(stderr, "rlogin: tcp/login: unknown service\n");
        exit(1);
}
```

Next the destination host is looked up with a gethostbyname call:

```
hp = gethostbyname(argv[1]);
if (hp == NULL) {
        fprintf(stderr, "rlogin: %s: unknown host\n", argv[1]);
        exit(2);
}
```

With this accomplished, all that is required is to establish a connection to the server at the requested host and start up the remote login protocol. The address buffer is cleared, then filled in with the internet address of the foreign host and the port number at which the login process resides:

```
memset((char *)&sin, `1Ø`,sizeof(sin));
memcpy hp->h_length), (char *)sin.sin_addr, (hp->h_addr;
sin.sin_family = hp->h_addrtype;
sin.sin_port = sp->s_port;
```

A socket is created, and a connection initiated:

```
s = socket(hp->h_addrtype, SOCK_STREAM, 0);
if (s < 0) {
       perror("rlogin: socket");
       exit(3);
}
 ...
if (connect(s, (char *)&sin, sizeof (sin)) < 0) {
       perror("rlogin: connect");
       exit(4);
}
```

The details of the remote login protocol will not be considered here.

CONNECTIONLESS SERVER PROCESS

While connection-based services are the norm, some services are based on the use of datagram sockets. One, in particular, is the rwho service which provides users with status information for hosts connected to a local area network. This service, while predicated on the ability to broadcast information to all hosts connected to a particular network, is of interest as an example usage of datagram sockets.

A user on any machine running the rwho server may find out the current status of a machine with the ruptime (1) program. The output generated is illustrated in Figure 13-4.

TBS

Figure 13-4. Ruptime Display Output

Status information for each host is periodically broadcast by rwho server processes on each machine. The same server process also receives the status information and uses it to update a database. This database is then interpreted to generate the status information for each host. Servers operate autonomously, coupled only by the local network and its broadcast capabilities.

The rwho server, in a simplified form, is pictured in Figure 13-5. There are two separate tasks performed by the server. The first task is to act as a receiver of status information broadcast by other hosts on the network. This job is carried out in the main loop of the program. Packets received at the rwho port are interrogated to insure they have been sent by another rwho server process, then are time stamped with their arrival time and used to update a file indicating the status of the host. When a host has not been heard from for an extended period of time, the database interpretation routines assume the host is down and indicates such on the status reports. This algorithm is prone to error since a server may be down while a host is actually up, but it works well in a LAN.

```
main()
{
        ...
        sp = getservbyname("who", "udp");
        net = getnetbyname("localnet");
        sin.sin_addr = inet_makeaddr(INADDR_ANY, net);
        sin.sin_port = sp->s_port;
        ...
        s = socket(AF_INET, SOCK_DGRAM, 0);
        ...
        bind(s, &sin, sizeof (sin));
        ...
        signal(SIGALRM, onalrm);
        onalrm();
        for (;;) {
                struct whod wd;
                int cc, whod, len = sizeof (from);

                cc = recvfrom(s, (char *)&wd, sizeof (struct whod), 0, &from,
         &len);
                if (cc <= 0) {
                        if (cc < 0 && errno != EINTR)
                                perror("rwhod: recv");
                        continue;
                }
                if (from.sin_port != sp->s_port) {
                        fprintf(stderr, "rwhod: %d: bad from port\n",
                                ntohs(from.sin_port));
                        continue;
                }
                ...
                if (!verify(wd.wd_hostname)) {
                        fprintf(stderr, "rwhod: malformed host name from %x\n",
                                ntohl(from.sin_addr.s_addr));
                        continue;
                }
                (void) sprintf(path, "%s/whod.%s", RWHODIR, wd.wd_hostname);
                whod = open(path, O.WRONLY|O.CREATE|O.TRUNCATE, 0666);
                ...
                (void) time(&wd.wd_recvtime);
                (void) write(whod, (char *)&wd, cc);
                (void) close(whod);
        }
}
```

Figure 13-5. Example of a Rwho Server.

The second task performed by the server is to supply information regarding the status of its host. This involves periodically acquiring system status information, packaging it up in a message and broadcasting it on the local network for other rwho servers to hear. The supply function is triggered by a timer and runs off a signal. Locating the system status information is somewhat involved, but not overly creative. Deciding where to transmit the resultant packet does, however, indicate some problems with the current protocol.

Status information is broadcast on the local network. For networks which do not support the notion of broadcast, another scheme must be used to simulate or replace broadcasting. One possibility is to enumerate the known neighbors (based on the status received). This, unfortunately, requires some bootstrapping information, as a server started up on a quiet network will have no known neighbors and thus never receive, or send, any status information. This is the identical problem faced by the routing table management process in propagating routing status information.

The standard solution, unsatisfactory as it may be, is to inform one or more servers of known neighbors and request that they always communicate with these neighbors. If each server has at least one neighbor supplied it, status information may then propagate through a neighbor to hosts which are not (possibly) directly neighbors. If the server is able to support networks which provide a broadcast capability, as well as those which do not, then networks with an arbitrary topology may share status information. (One must, however, be concerned about "loops." That is, if a host is connected to multiple networks, it will receive status information from itself. This can lead to an endless, wasteful, exchange of information.)

The second problem with the current scheme is that the rwho process services only a single local network, and this network is found by reading a file. It is important that software operating in a distributed environment not have any site-dependent information compiled into it. This would require a separate copy of the server at each host and make maintenance a problem. 4.2BSD attempts to isolate host-specific information from applications by providing system calls which return the necessary information. (An example of such a system call is the gethostname(2) call which returns the host's "official" name.)

IPC PROGRAMMING TECHNIQUES

A number of facilities have yet to be discussed. For most users of the ipc the mechanisms already described will suffice in constructing distributed applications. However, others will find need to utilize some of the features which we consider in this section.

OUT OF BAND DATA

The stream socket abstraction includes the notion of "out of band" data. Out of band data is a logically independent transmission channel associated with each pair of connected stream sockets. Out of band data is delivered to the user independently of normal data along with the SIGURG signal. (On CTIX systems, this is equivalent to SIGUSR1). In addition to the information passed, a logical mark is placed in the data stream to indicate the point at which the out of band data was sent. The remote login and remote shell applications use this facility to propagate signals from between client and server processes. When a signal is expected to flush any pending output from the remote process(es), all data up to the mark in the data stream is discarded.

The stream abstraction defines that the out of band data facilities must support the reliable delivery of at least one out of band message at a time. This message may contain at least one byte of data, and at least one message may be pending delivery to the user at any one time. For communications protocols which support only in-band signaling (that is, the urgent data is delivered in sequence with the normal data) the system extracts the data from the normal data stream and stores it separately. This allows users to choose between receiving the urgent data in order and receiving it out of sequence without having to buffer all the intervening data.

To send an out of band message the SOF_OOB flag is supplied to a send or <u>sendto</u> calls, while to receive out of band data SOF_OOB should be indicated when performing a <u>recvfrom</u> or <u>recv</u> call. To find out if the read pointer is currently pointing at the mark in the data stream, the SIOCATMARK ioctl is provided:

```
ioctl(s, SIOCATMARK, &yes);
```

If <u>yes</u> is a 1 on return, the next read will return data after the mark. Otherwise (assuming out of band data has arrived), the next read will provide data sent by the client prior to transmission of the out of band signal. The routine used in the remote login process to flush output on receipt of an interrupt or quit signal is shown in Figure 13-6.

```
oob()
{
        int out = 1+1;
        char waste[BUFSIZ], mark;

        signal(SIGURG, oob);
        /* flush local terminal input and output */
        ioctl(1, TIOCFLUSH, (char *)&out);
        for (;;) {
                if (ioctl(rem, SIOCATMARK, &mark) < 0) {
                        perror("ioctl");
                        break;
                }
                if (mark)
                        break;
                (void) read(rem, waste, sizeof (waste));
        }
        recv(rem, &mark, 1, SOF_OOB);
        ...
}
```

Figure 13-6. Flushing Terminal I/O on Receipt of Out of Band Data.

SIGNAL AND PROCESS GROUPS

Because of the existence of the SIGURG signal, each socket has an associated process group (just as is done for terminals). This process group is initialized to the process group of its creator, but may be redefined at a later time with the SIOCSPGRP ioctl:

```
ioctl(s, SIOCSPGRP, &pgrp);
```

A similar ioctl, SIOCGPGRP, is available for determining the current process group of a socket.

PSEUDO TERMINALS

Many programs will not function properly without a terminal for standard input and output. Since a socket is not a terminal, it is often necessary to have a process communicating over the network do so through a pseudo terminal. A pseudo terminal is actually a pair of devices, master and slave, which allow a process to serve as an active agent in communication between processes and users. Data written on the slave side of a pseudo terminal is supplied as input to a process reading from the master side. Data written on the master side is given the slave as input. In this way, the process manipulating the master side of the pseudo terminal has control over the information read and written on the slave side.

The remote login server uses pseudo terminals for remote login sessions. A user logging in to a machine across the network is provided a shell with a slave pseudo terminal as standard input, output, and error. The server process then handles the communication between the programs invoked by the remote shell and the user's local client process. When a user sends an interrupt or quit signal to a process executing on a remote machine, the client login program traps the signal, sends an out of band message to the server process who then uses the signal number, sent as the data value in the out of band message, to perform a _kill_(2) on the appropriate process group.

INTERNET ADDRESS BINDING

Binding addresses to sockets in the Internet domain can be fairly complex. Communicating processes are bound by an association. An association is composed of local and foreign addresses, and local and foreign ports. Port numbers are allocated out of separate spaces, one for each Internet protocol. Associations are always unique. That is, there may never be duplicate <protocol, local address, local port, foreign address, foreign port> tuples.

The bind system call allows a process to specify half of an association,

<local address, local port>

while the connect and accept primitives are used to complete a socket's association. Since the association is created in two steps, the association uniqueness requirement indicated above could be violated unless care is taken. Further, it is unrealistic to expect user programs to always know proper values to use for the local address and local port since a host may reside on multiple networks and the set of allocated port numbers is not directly accessible to a user.

To simplify local address binding the notion of a "wildcard" address has been provided. When an address is specified as INADDR_ANY (a manifest constant defined in **sys/in.h**), the system interprets the address as "any valid address." For example, to bind a specific port number to a socket, but leave the local address unspecified, the following code might be used:

```
#include <sys/types.h>
#include <sys/in.h>
 ...
struct sockaddr_in sin;
 ...
s = socket(AF_INET, SOCK_STREAM, 0);
sin.sin_family = AF_INET;
sin.sin_addr.s_addr = INADDR_ANY;
sin.sin_port = MYPORT;
bind(s, (char *)&sin, sizeof (sin));
```

Sockets with wildcarded local addresses may receive messages directed to the specified port number, and addressed to any of the possible addresses assigned a host. For example, if a host is on a networks 46 and 10 and a socket is bound as above, then an accept call is performed, the process will be able to accept connection requests which arrive either from network 46 or network 10.

In a similar fashion, a local port may be left unspecified (specified as zero), in which case the system will select an appropriate port number for it. For example:

```
sin.sin_addr.s_addr = MYADDRESS;
sin.sin_port = 0;
bind(s, (char *)&sin, sizeof (sin));
```

The system selects the port number based on two criteria. The first is that ports numbered 0 through 1023 are reserved for privileged users (that is, the super user). The second is that the port number is not currently bound to some other socket. In order to find a free port number in the privileged range the following code is used by the remote shell server:

```
struct sockaddr_in sin;
 ...
lport = IPPORT_RESERVED - 1;
sin.sin_addr.s_addr = INADDR_ANY;
 ...
for (;;) {
        sin.sin_port = htons((u_short)lport);
        if (bind(s, (caddr_t)&sin, sizeof (sin)) >= 0)
                break;
        if (errno != EADDRINUSE && errno != EADDRNOTAVAIL) {
                perror("socket");
                break;
        }
        lport--;
        if (lport == IPPORT_RESERVED/2) {
                fprintf(stderr, "socket: All ports in use\n");
                break;
        }
}
```

The restriction on allocating ports was done to allow processes executing in a "secure" environment to perform authentication based on the originating address and port number.

In certain cases the algorithm used by the system in selecting port numbers is unsuitable for an application. This is because of associations being created in a two step process. For example, the Internet file transfer protocol, ftp, specifies that data connections must always originate from the same local port. However, duplicate associations are avoided by connecting to different foreign ports. In this situation the system would disallow binding the same local address and port number to a socket if a previous data connection's socket were around. To override the default port selection algorithm then an option call must be performed prior to address binding:

```
setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *)Ø,
 Ø);
bind(s, (char *)&sin, sizeof (sin));
```

With the above call, local addresses may be bound which are already in use. This does not violate the uniqueness requirement as the system still checks at connect time to be sure any other sockets with the same local address and port do not have the same foreign address and port (if an association already exists, the error EADDRINUSE is returned).

Local address binding by the system is currently done somewhat haphazardly when a host is on multiple networks. Logically, one would expect the system to bind the local address associated with the network through which a peer was communicating. For instance, if the local host is connected to networks 46 and 1Ø and the foreign host is on network 32, and traffic from network 32 were arriving via network 1Ø, the local address to be bound would be the host's address on network 1Ø, not network 46. This unfortunately, is not always the case. For reasons too complicated to discuss here, the local address bound may be appear to be chosen at random. This property of local address binding will normally be invisible to users unless the foreign host does not understand how to reach the address selected. (For example, if network 46 were unknown to the host on network 32, and the local address were bound to that located on network 46, then even though a route between the two hosts existed through network 1Ø, a connection would fail.)

BROADCASTING AND DATAGRAM SOCKETS

By using a datagram socket it is possible to send broadcast packets on many networks supported by the system (the network itself must support the notion of broadcasting; the system provides no broadcast simulation in software). Broadcast messages can place a high load on a network since they force every host on the network to service them. Consequently, the ability to send broadcast packets has been limited to the super user.

To send a broadcast message, an Internet datagram socket should be created:

```
        s = socket(AF_INET, SOCK_DGRAM, 0);
```

and at least a port number should be bound to the socket:

```
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = MYPORT;
        bind(s, (char *)&sin, sizeof (sin));
```

Then the message should be addressed as:

```
        dst.sin_family = AF_INET;
        dst.sin_addr.s_addr = INADDR_ANY;
        dst.sin_port = DESTPORT;
```

and, finally, a sendto call may be used:

```
  sendto(s, buf, buflen, 0, &dst, sizeof (dst));
```

Received broadcast messages contain the senders address and port. (Datagram sockets are anchored before a message is allowed to go out.)

USING SPECIAL SIGNALS

There are two signals that can be used in conjunction with the interprocess communication facilities. The SIGURG signal is associated with the existence of an "urgent condition." SIGUSR1 is currently supplied a process when out of band data is present at a socket. If multiple sockets have out of band data awaiting delivery, a select call may be used to determine those sockets with such data.

Preventing Zombies

Berkeley 4.3BSD uses the signal SICCHLD to reap child processes after exiting, thus preventing "zombie" processes from accumulating. However, in CTIX Internetworking it is recommended to obviate the need for reaping child processes by using the following signal set argument:

```
signal(SIGCLD, SIG_IGN);
```

Place the argument in the initialization segment, before you fork the child process. If the child dies before you make(1) the argument, the child may have to be reaped as described in the 4.2BSD Interprocess Communications Primer.

TYPICAL TCP/IP PROCESS

A state diagram for establishing a TCP stream socket (SOCK_STREAM) is shown in Figure 13-7.

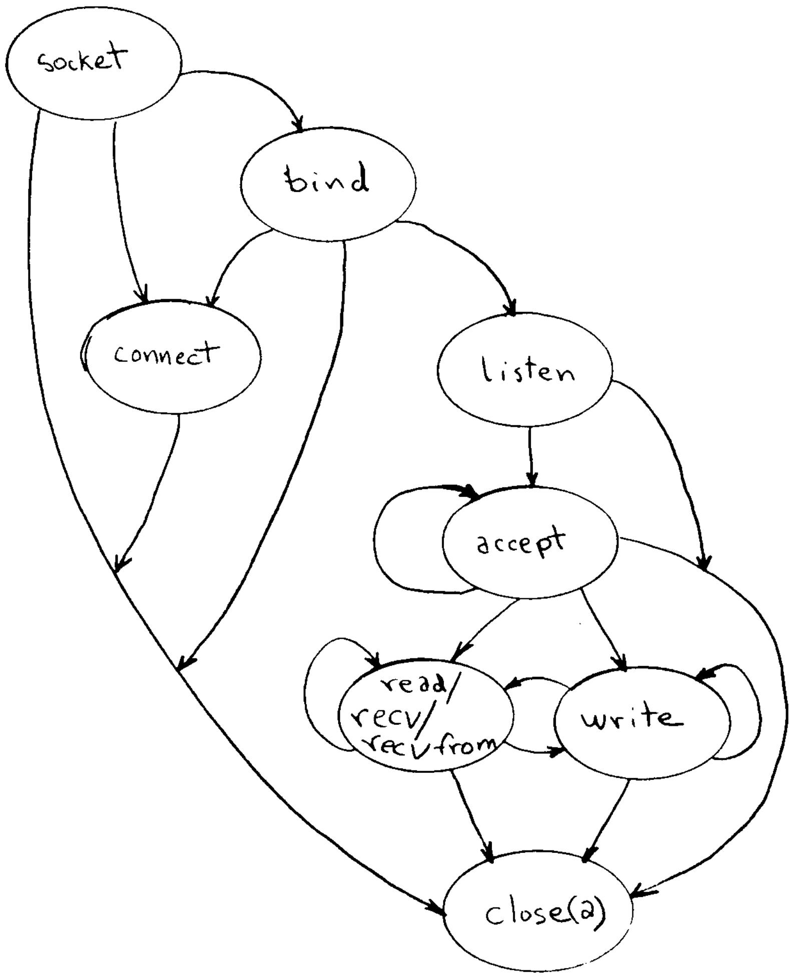


Figure 13-7. State Diagram for TCP Stream Socket.

The calling program issues a socket call to AF_INET (Address Family Internet) to get access to the net.

TCP returns a file descriptor. If the calling program wishes a particular circuit, it looks in **/etc/services**, using a getservent library call, for the port assignment for the service it requires and issues a bind call to that address number. For example, the service ftp is commonly assigned port 21.

The program gives the port of origin and the the full destination address (family, port, host address).

(Well known services have permanently assigned ports.) TCP ports are logical ports associated with sockets, not physical communications ports. (See **/etc/services** for standard port assignments.) TCP allocates the port to the calling program. If the program does not ask to bind a port, TCP assigns one when it receives the connect call.

The calling program then issues a connect call by identifying its own full address and the full destination address. IP addresses should be retrieved through the gethostent call.

When the connection completes, the connection is established, the calling program writes and/or reads.

If the write exceeds the maximum packet size, TCP breaks it up into separate packets.

The receiving server does much the same thing at the other end, a socket call and a bind call. The receiving server then makes a listen call and an accept, which says, "Wait for a call, and if one comes in, give it to me." In the example of the ftp server, when ftp accepts a call, it spins off a child process which takes over the ftp operations. The original ftp demon goes back to waiting for another call. The child process takes over the new connection.

The TCP protocol on each machine keeps track of multiple concurrent sessions between machines by assigning numbered ports to the sessions. For example, session A is between machine 1.1, port 4 and machine 1.2, port 2; while session B is between machine 1.1, port 5 and machine 1.2, port 3, and so on.

TYPICAL UDP PROCESS

A state diagram for establishing a UDP datagram socket (SOCK_DGRAM) is shown in Figure 13-8.

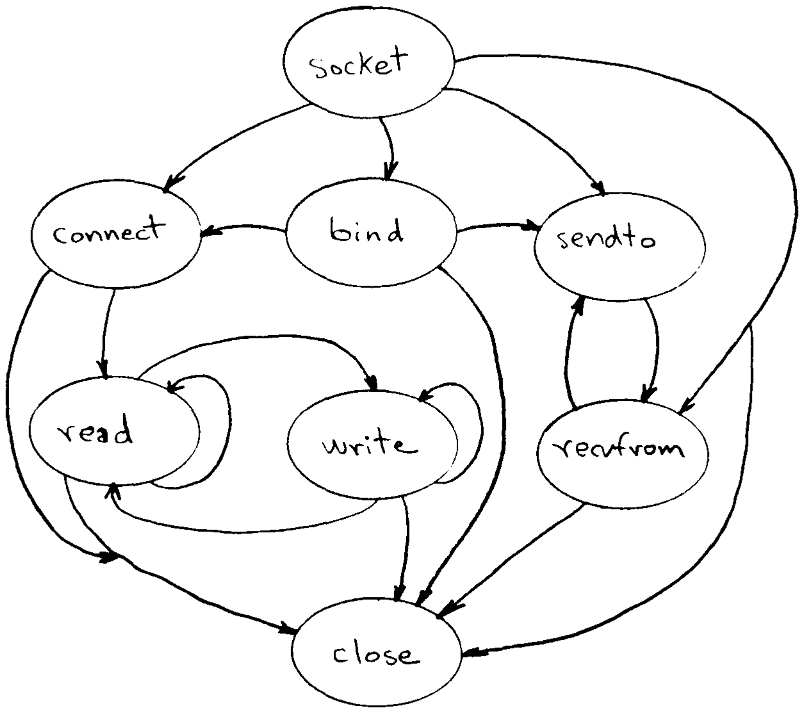


Figure 13-8. State Diagram for UDP Datagram Socket.

PROGRAMMING HINTS

ADDING AND CHECKING FOR SERVICES

When you need to add a new service, you must add its name to the **/etc/services** file. In your program you can issue a call to look into **/etc/services** to make sure it is on the system, before using it in your program. See getservent(3N) and Chapter 8, "Network Management."

ERROR HANDLING IN PROGRAMMING

(See Chapter 14, "Troubleshooting.")

14 TROUBLESHOOTING

CTIX has approximately 70 generic error codes. Generally, each system service maps it error condition to these codes. Some errors have been added for TCP/IP. (See the appropriate CTIX Operating System Manual.)

MIGHTYFRAME OPERATOR MESSAGES

There is no fixed set system console on the MightyFrame. If an operator response is required, the system looks for any active terminal. System error messages are logged in **/etc/log/confile.**

ERROR HANDLING IN PROGRAMMING

You should check for error returns after every system call. Most calls have one or more error returns. These errors are described in the call description in the appropriate CTIX Operating System Manual. An error condition is indicated by an otherwise impossible returned value. This is almost always **-1.** An error number is made available in the external variable **errno. Errno** is set only when an error incurred and is not cleared on successful calls, therefore test it only after an error has been indicated. Use perror(3C) to print error messages. (For a complete list of the 88 or more error returns found in **errno.h,** see intro(2) in the appropriate CTIX Operating System Manual. Note that the error numbers are subject to change.)

ERROR MESSAGES

DATAGRAM SOCKET ERROR MESSAGES

If information at the sending node indicates that the message cannot be delivered, for instance, when a network is unreachable, the call returns - 1 and the global value errno will contain an error number.

ALPHABETICAL LIST OF ERROR MESSAGES

The following is an alphabetical list of internetworking error message which can be displayed on the terminal or console. The protocol or operation during which the error is likely to occur is given at the beginning of the explanation.

EADDRINUSE	TCP and UDP. An attempt was made to create a socket with a port which has already been allocated.
EADDRNOTAVAIL	TCP and UDP. An attempt was made to create a socket with a network address for which no network interface exists.
ECONNREFUSED	Socket Connection. The host refused service for some reason. This error is usually caused by a server process not being present at the requested name.
EHOSTDOWN	(See ENETDOWN)
EHOSTUNREACH	(See ENETUNREACH)
ECONNREFUSED	TCP. The remote peer actively refuses connection establishment (usually because no process is listening to the port).

ECONNRESET	TCP. The remote peer forced the session to be closed.
EISCONN	IP and UDP. An attempt was made to establish a connection on a socket that already has one or an attempt was made to send a datagram with the destination address specified and the socket is already connected.
EISCONN	TCP. An attempt was made to establish a connection on a socket that already has one.
ENETDOWN or EHOSTDOWN	Socket Connection. Status information received by the client host from the underlying communication services indicates the net or the remote host is down.
ENETUNREACH or EHOSTUNREACH	Socket Connection. These operational errors can occur either because the network or host is unknown (no route to the host or network is present) or because status information to that effect has been delivered to the client host by the underlying communication services.
ENOBUFS	TCP, IP, and UDP. Any Socket Operation. The system lacks sufficient memory for an internal data structure.
ENOTCONN	UDP. An attempt was made to send a datagram, but no destination address is specified, and the socket has not been connected.

EPROTONOSUPPORT	
	Creating a Socket. Unknown protocol or protocol not supported.
EPROTOTYPE	Creating a Socket. Socket type request has no supporting protocol.
ETIMEDOUT	Socket Connection. After failing to establish a connection during a period of time (excessive retransmissions), the system decided there was no point in retrying any more. The cause for this error is usually that the remote host is down or that problems in the network resulted in transmissions being lost.

(For additional networking error messages, see MegaFrame CTIX Operating System Manual.)

APPENDIX A TELNET COMMAND SUMMARY

This appendix contains a brief summary of the commands available within telnet. Telnet is described in Chapter 3, "Using the Virtual Terminal."

Command names are in **bold face.** Arguments are underlined.

AO	send abort output
AYT	send are you there
BREAK	send break
EC	send erase character
EL	send erase line
IP	send interrupt process
SYNCH	send synch
crmod	toggle cr mapping mode
close	close the connection
escape	change the escape character
help	help
open node	establish a connection
options	toggle option display
quit	exit telnet
status	show telnet status

APPENDIX B   FTP COMMAND SUMMARY

This appendix contains a brief summary of the commands available within ftp. Command names are in **bold face**. Arguments are underlined. Optional arguments are enclosed in square brackets.

!	invoke a shell
append local-file [remote-file]	append a file
ascii	use ascii file transfer mode
bell	toggle bell mode
binary	use binary file transfer mode
bye	exit ftp
cd remote-directory	change remote directory
close	close current connection
copy host1:file1 host2:file2	copy file
delete remote-file	delete file
debug	toggle debug mode
dir [remote-directory] [local-file]	list directory
get remote-file [local-file]	receive a file
form	set file transfer format
glob	toggle wild card expansion
hash	toggle hash display mode
help [command-name]	help
lcd directory	change local directory
ls remote-directory local-file	list directory
mdelete remote-files	delete multiple files
mdir remote-files local-file	directory for multiple files
mget remote-files	receive multiple files
mkdir directory-name	make directory
mls remote-files local-file	directory for multiple files
mput local-files	send multiple files
open host [port]	open a connection
prompt	toggle prompt mode
put local-files remote-file	send a file
pwd	show remote directory
quit	exit ftp
quote arg1 arg2 ...	send a string
recv remote-file [local-file]	receive a file
remotehelp [command-name]	remote help
rename remote-file remote-file	rename a file
rmdir directory-name	delete a directory
send local-file remote-file	send a file
sendport	toggle send port mode
status	display ftp status
trace	toggle trace mode
type [type-name]	set file transfer type
user user-name [password] [account]	login to a remote machine
verbose	toggle verbose mode
? [command]	help

APPENDIX C   SAMPLE /etc/rc FILE

```
TZ=`cat /etc/TZ`; export TZ
PATH=/bin:/usr/bin:/etc:/usr/local/bin; export PATH

:       set UUCP node name here
setuname -n network
:       uncomment the following line to set the internet address.
: /etc/setaddr

if [ ! -f /etc/mnttab ]
then
        > /etc/mnttab
        devnm / | setmnt
fi
# coming from single going to multi
set `who -r`
if [ \( "$9" = "S" \) -a \( "$7" = "2" -o "$7" = "3" \) ]
then
        : put mounts in mountable
        /etc/mountable
        rm -f /usr/adm/acct/nite/lock*
        /usr/lib/ex3.9preserve -
        # BACCT (marker for scripts)
#       /bin/su - adm -c /usr/lib/acct/startup
#       echo process accounting started
        # EACCT
        # BERR (marker for scripts)
        /etc/errdead -ae
        echo errdemon started
        # EERR
        # BSAR (marker for scripts)
#       /bin/su - sys -c "/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d` &"
        # ESAR
        test -f /usr/adm/sulog &&  mv /usr/adm/sulog /usr/adm/OLDsulog
        test -f /usr/lib/cron/log &&  mv /usr/lib/cron/log /usr/lib/cron/OLDlog
        > /usr/lib/cron/log
        /etc/cron
        echo cron started
        /etc/update 30&
        echo update started
        mv /usr/tmp/oas.* /usr/preserve/. 2>/dev/null
        rm -rf /tmp/*
        rm -rf /usr/tmp/*
        mv /usr/preserve/oas.* /usr/tmp/. 2>/dev/null
        rm -f /usr/spool/uucp/LCK*
#       To invoke the lp spooler, uncomment the following lines, and comment
#       the lines concerning lpr/lpd.
        # BLP (marker for scripts)
        rm -f /usr/spool/lp/SCHEDLOCK
        /usr/lib/lpsched
        echo "LP Scheduler Started"
        # ELP
        # BLPR (marker for scripts)
#       rm -f /usr/spool/lpd/lock
#       /usr/lib/lpd
        # ELPR
        /etc/enpstart network && echo Ethernet Processors Started


        # BNET (marker for scripts)
        DEMONS=' rlogind rshd telnetd uucpd'
        echo Starting Network Demons: " \c"
        for i in $DEMONS
        do
                /etc/$i
                echo $i " \c"
        done
        echo
        # ENET
        # Set attributes for parallel line printer
        #lpset -i4 -c132 -166
fi
```

APPENDIX D SAMPLE /etc/hosts FILE

```
#       List of hosts on the CT Engineering Net.
#       To add a new host, mail it to Tom Faulhaber who will
#       distribute it to the net.
#
#       The Engineering Ethernet is defined to be network #3 for
#       no apparent reason.  Other network numbers will be created as
#       needed.  See the /etc/networks file for more information.

# Machines in the Distributed Unix Group

3.38    network                 # Network Development Mighty - Tom Faulhaber
3.12    tom-src net3            # Network Source Machine - Tom Faulhaber
3.10    tom-test net2           # Network Test Machine - Tom Faulhaber
3.14    andrew                  # Network Development - Andrew Knutsen

# MiniFrame & TWS Machines
3.16    mifa                    # MiniFrame Group Development - Rob Farnum
3.28    mini+                   # Rob Farnum's miniframe plus.

# MightyFrame Team Machines

3.20    moose2 dateserver       # MightyFrame Development - Chris Wagner.
3.18    mitimouse               # MightyFrame Test - Bruce Beare.
3.44    raymini                 # Ray Bloom's MiniFrame.

# General Engineering (Non-development) Machines

3.22    mifb                    # MiniFrame Hardware - Jamie Riotto.
3.24    mifc                    # Engineering Management - Carl Rigg.

# Unix Communications Development

3.26    TQmini                  # Comm Group MiniFrame - Hemant
3.32    BTJmini                 # Comm Group MiniFrame - Bent Jensen.
3.42    vbmini                  # V.B.'s MiniFrame.

# Commands and Languages

3.34    jjs                     # John Sancho's MiniFrame.
3.36    wiltse                  # Wiltse Carpenter's MiniFrame.

# Design Verification.

3.46    dv1                     # Design Verification Mini #1, T.S. Reddy
3.48    dv2                     # Design Verification Mini #2, T.S. Reddy

# Network #1 is Tom's asynchronous serial network.
1.1     serial1                 # Serial test station #1 (tom-test).
1.2     serial2                 # Serial test station #2 (tom-src).
# 1.10  tom-test net2           # Network Test Machine - Tom Faulhaber

127.1   loopback                # generic loopback port.
```

APPENDIX E   SAMPLE /etc/networks FILE

```
#       This file describes the various networks and gateways
#       used at CT.

#       If you add a new gateway you should mail it to Tom Faulhaber
#       who will update the network.

Serial-Net          1       # Tom's Serial Test Network.

Engineering-Net     3       # Main CTIX Development Ethernet.

X25-Net             5       # X.25 Based Network (Comm Group).

DV-Net              7       # Comm Design Verification's Ethernet.

Loopback            127
```

APPENDIX F SAMPLE /etc/gethosts FILE

```
UPD_UUCP=TRUE
UPD_LP=FALSE
UPD_RWHO=TRUE
UPD_HOSTS=TRUE
UPD_MAILX=TRUE


NODE=tom-src

error="fgrep -v 'bytes received' | egrep '^5|Unknown host'"

cd /tmp
/usr/local/bin/ftp -n <<eof 2>&1 | eval $error
open $NODE
user ftp ftp
verbose
cd gethosts
get hosts
get networks
get mailx.rc
bye
eof

if [ $? -eq 0 ]
then
        echo Can\'t get current hosts file -- FAILED.
        exit 1
fi

if [ ! -s /tmp/hosts ]
then
        echo Hosts file is zero length -- FAILED.
        rm -f /tmp/hosts
        exit 1
fi

cmp -s /tmp/hosts /etc/hosts

if [ $? -eq 1 ]
then
        mv /etc/hosts /etc/OLDhosts
        cp /tmp/hosts /etc/hosts
        chown bin /etc/hosts
        chgrp bin /etc/hosts
        chmod 644 /etc/hosts
        echo Updating file /etc/hosts from $NODE
else
        echo Host file is up-to-date
fi
rm -f /tmp/hosts

cmp -s /tmp/networks /etc/networks

if [ $? -eq 1 ]
then
        mv /etc/networks /etc/OLDnetworks



        cp /tmp/networks /etc/networks
        chown bin /etc/networks
        chgrp bin /etc/networks
        chmod 644 /etc/networks
        echo Updating file /etc/networks from $NODE
else
        echo Network file is up-to-date
fi
rm -f /tmp/networks

# This part of gethosts updates the various files based on variables
# set at installation time.
```

SAMPLE /etc/gethosts FILE (continued)

```
if [ "$UPD_UUCP" = "TRUE" ]
then
        # update L.sys
        cp /usr/lib/uucp/L.sys /usr/lib/uucp/OLDL.sys
        fgrep -v INET /usr/lib/uucp/L.sys >/tmp/L1  # remove names to avoid dup

        for i in `awk ' $1 != "#" { print $2 }' /etc/hosts`
        do
                echo "$i Any INET 9600 nuucp" >> /tmp/L1
        done
        mv /tmp/L1 /usr/lib/uucp/L.sys
        chmod 400 /usr/lib/uucp/L.sys
        chown uucp /usr/lib/uucp/L.sys
        chgrp bin /usr/lib/uucp/L.sys


        # update /usr/spool/uucppublic/.rhosts

        mv -f /usr/spool/uucppublic/.rhosts /usr/spool/uucppublic/OLDrhosts
        cat </dev/null >/usr/spool/uucppublic/.rhosts
        for i in `awk ' $1 != "#" { print $2 }' /etc/hosts`
        do
                echo $i uucp >>/usr/spool/uucppublic/.rhosts
        done
        chown nuucp /usr/spool/uucppublic/.rhosts
        chgrp bin /usr/spool/uucppublic/.rhosts
        chmod 400  /usr/spool/uucppublic/.rhosts
        echo UUCP updated
fi

if [ "$UPD_LP" = "TRUE" ]
then
        mv /usr/spool/lp/.rhosts /usr/spool/lp/OLDrhosts
        cat </dev/null >/usr/spool/lp/.rhosts
        for i in `awk ' $1 != "#" { print $2 }' /etc/hosts`
        do
                echo $i lp >>/usr/spool/lp/.rhosts
        done
        chown lp /usr/spool/lp/.rhosts
        chgrp bin /usr/spool/lp/.rhosts
        chmod 400  /usr/spool/lp/.rhosts
        echo LP updated
fi
```

SAMPLE /etc/gethosts FILE
(continued)

```
# check the rwho stuff to make sure there are no discontinues machines.

if [ "$UPD_RWHO" = "TRUE" ]
then
        for i in `awk ' $1 != "#" { print $2 }' /etc/hosts`
        do
                echo whod.$i  >>/tmp/syslist
        done
        cd /usr/spool/rwho
        for i in *
        do
                fgrep -x $i /tmp/syslist >/dev/null 2>&1
                if [ $? -ne 0 ]
                then
                        rm -f $i
                        echo Removed /usr/spool/rwho/$i
                fi
        done
        rm -f /tmp/syslist
        echo Remote Who data updated.
fi

# update the /usr/hosts directory

if [ "$UPD_HOSTS" = "TRUE" ]
then
        rm -f /usr/hosts/*
        /etc/mkhosts
        echo /usr/hosts directory updated
fi

# update the /usr/lib/mailx directory

if [ "$UPD_MAILX" = "TRUE" ]
then
        if [ ! -f /usr/lib/mailx/local.rc ]
        then
                cat /dev/null >/usr/lib/mailx/local.rc
                chown bin /usr/lib/mailx/local.rc
                chgrp bin /usr/lib/mailx/local.rc
                chmod 644 /usr/lib/mailx/local.rc
        fi

        if [ ! -f /usr/lib/mailx/mailx.rc ]
        then
                cat /dev/null >/usr/lib/mailx/mailx.rc
        fi

        cmp -s /tmp/mailx.rc /usr/lib/mailx/mailx.rc

        if [ $? -eq 1 ]
        then
                mv /usr/lib/mailx/mailx.rc /usr/lib/mailx/OLDmailx.rc
                mv /tmp/mailx.rc /usr/lib/mailx/mailx.rc
                chown bin /usr/lib/mailx/mailx.rc


                chgrp bin /usr/lib/mailx/mailx.rc
                chmod 644 /usr/lib/mailx/mailx.rc
                echo /usr/lib/mailx directory updated
        else
                echo /usr/lib/mailx directory is up-to-date
        fi
fi
rm -f /tmp/mailx.rc
```

APPENDIX G INTERNET ADDRESSES

Theoretically, an internet address is available for any machine in the world implementing TCP/IP networking protocols. This address is provided by DARPA addressing scheme. These numbers are assigned by the University of Southern California Sciences Institute.

An internet address is made up of an address class identifier, a network number, and a local host address number. The address class identifier is either Ø, 1Ø, or 11Ø for Class A, B, or C, respectively. The network number identifies a unique physical network in the internet. The local address carries information to address a host in the network identified by the network number.

The internet address is a 32-bit quantity formatted differently in three types, or classes, to accommodate different network size configurations. Class A allocates a 7-bit network number and a 24-bit local address. Class B allocates a 14-bit network number and a 16-bit local address. Class C allocates a 21-bit network number and an 8-bit local address. Figure G-1 gives the formats of the address types.

This system provides a unique address for the entire statistical distribution that might be expected in the total population of networks using this address system. There would be a smaller number of large network, having many nodes (Class A), and a larger number of small networks, consisting of a lesser number of nodes (Class C), and a medium number of network made up of a medium number of nodes (Class B).

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|   NETWORK   |                Local Address                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                       Class A Address
```

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 0|           NETWORK             |        Local Address      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                       Class B Address
```

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 0|                    NETWORK              | Local Address |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                       Class C Address
```

Figure G-1. Classes of Internet Addresses

Because each network can have a particular address format and length (class A, B, or C), the IP maps between the internet local addresses and the actual address format used in the particular network. For information on the mapping relationship between address fields used in the Internet Protocol and specific network such as ARPANET, see RFC 814 in the Internet Protocol Transition Workbook.

GLOSSARY

alias. An alias is an alternate host name, which can be created as a convenience in addressing a host on a local network whose unique primary name is long and/or complicated.

ARP. Address Resolution Protocol is used by Ethernet for address mapping.

ARPA. Now called DARPA, stands for Defense Advanced Research Projects Agency. ARPANET is the network based on the work sponsored by this agency. See also **DDN**.

block. A block (noun) is usually 1024 bytes.

broadcast network. A broadcast network is one that "broadcasts" all transmissions instead of from point to point. Each node then "grabs" the transmissions intended for them. For example, Ethernet broadcasts down its bus.

BSD. Berkeley Software Distribution.

bus. A set of parallel signals implemented in hardware in a standard manner so that multiple devices can access it and communicate over it. The MightyFrame uses the VME bus. The MegaFrame uses the Multibus indirectly through an adapter, the MiniFrame uses a proprietary bus.

communications server. A communications server is a user-shareable system service installed on a specific machine and is accessed by the user explicitly to provide a communications link to an specific outside service. An example of a communications server is an X.29 pad.

connection. A connection is a logical communication path identified by a pair of sockets.

CTIX machine. A MightyFrame, MiniFrame, or MegaFrame or other computer that run the CTIX operating system.

DARPA. Department of Defense Advanced Research Project Agency, formerly called ARPA. This agency sponsored the network architecture research project upon which ARPANET is based. ARPANET is a large governmental internetwork, called the Internet, part of which is the Defense Data Network (DDN). See also **DDN and Internet.**

data link level. Data link level is the communications protocol for the physical media-link used to transport the data.

datagram. A datagram is a message sent in a packet switched computer communications network. The message made up of source and destination addresses and the data itself. The datagram model implies that no connection, such as a virtual circuit is needed, to send them and that they are not required to be delivered reliably or in sequence. See also **packet.**

DDN. Defense Data Network. The Defense Data Network (DDN) is a set of communications capabilities which links together computer systems within the Department of Defense (DoD). The DDN allows users of these computer systems to send mail and files between systems and to access other computers on the network in interactive terminal sessions. The DDN is part of the DARPA Internet. See **Internet.**

DDN Network Gateway. The DDN Network Gateway is a special product that allows users of the MegaFrame running the CTIX operating system to access the Defense Data Network and communicate with other machines on the network. For more information see DDN MegaFrame Reference Manual.

demon. A demon is a CTIX system service. It is a program that is active in the background but not connected to a terminal.

destination. The destination address, an internet header field.

destination Address. The destination address, usually the network and host identifiers.

download. To download is to move an executable server from a CTIX file that would usually run in the processor's main memory to an intelligent circuit board that on the processor bus. The intelligent board then executes the server.

ENP. Ethernet Node Processor.

Flags. Flags is an internet header field carrying various control flags.

flow control. Flow control is the function and process of regulating the traffic and amount of data between flowing nodes so that neither node is sent more data than it can handle at a given time.

gateway. A software service installed at a switching node that connects two or more networks, especially if they use different protocols. A gateway provides CTIX internetworking with an extended logical network by transparently attaching one or more physical networks. See also **communications server.**

header. A header is the control information at the beginning of a message, segment, datagram, fragment, packet or block of data.

host. A host is a computer. In particular a source or destination of messages from the point of view of the communication network.

ICMP. Internet Control Message Protocol. ICMP is used by a gateway or destination host to communicate with a source host, for example, to report an error in datagram processing. ICMP, uses the basic support of IP as if ICMP were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

Identification. Identification is an Internet Protocol field. This identifying value assigned by the sender aids in assembling the fragments of a datagram.

install. Install means to move the executable files from the distribution media to the system disk. In context, install can also mean to perform all the steps necessary to make a server or protocol operative.

Internet. The Internet (spelled with initial capitalization) is the DARPA Internet System. See **DARPA.**

Internet Address. Internet Address is a source or destination address specific to the host level. It consists of a four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.

Internet datagram. An internet datagram is the unit of data exchanged between a pair of internet modules (includes the internet header).

internet module. An internet module is an instance or individual implementation of the Internet Protocol, residing at a local host or gateway.

Internet Protocol. Internet Protocol (IP) is the network level protocol used by CTIX internetworking.

internetwork. An internetwork is a supernetwork made up of two or more networks able to communicate with each other through gateways. See **gateway.**

IP. See **Internet Protocol.**

layer. A layer is a network function or set of related network functions that forms an autonomous functional block in the superset of network architectural functions. This method of partitioning the necessary network functions allows each layer to interface transparently to adjoining layers and thereby provides a method of making network components more manageable.

load. Load means to execute a command which causes a loadable driver, demon, or system service to be called into memory and become active.

Local Address. The Local Address the address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.

local packet. A local packet is the unit of transmission within a local network.

machine. A machine is a host computer. The use of this term is similar to "host," and "node," but "machine" connotates the machine-specific or hardware aspects of the host computer, whereas "node" connotates the logical aspects of a network host. Host connotates the relationship of the local node machine to application systems and remote hosts.

MegaFrame CTIX X.25 Interface. The MegaFrame CTIX X.25 Interface is a loadable server for the MegaFrame that provides in interface to an X.25 PDN for the CTIX TCP/IP-X.25 Interface and other applications. Its counterpart on the MightyFrame is the MightyFrame X.25 Network Gateway.

mbuf. A memory buffer is a unit of memory usage used in the Memory Usage display of netman. Each mbuf uses 128 bytes of memory

MightyFrame CTIX TCP/IP-X.25 Interface. The MightyFrame CTIX TCP/IP-X.25 Interface is an optional internetworking product that allows TCP/IP to access the MightyFrame X.25 Network Gateway.

module. A module is an implementation, usually in software, of a protocol or other procedure.

network. A network is a collection of computer nodes able to communicate with each other.

network interface. A network interface is the hardware and driver software that connects a host to a physical network.

octet. An octet is an eight bit byte.

OSI (Open Systems Interconnection). OSI is a standard of the ISO. This standard attempts to provide for consistent hardware and software interfaces among network products. OSI and other standard setters such as IBM and the National Bureau of Standards generally divide network architecture into seven layers: physical, link, network, transport, session, presentation, and application.

packet. A packet is a package of data with a header which may or may not be logically complete. More often a physical packaging than a logical packaging of data.

packet-switching. Packet switching is a method used by certain networks to transfer data. The data is transmitted in small segments called packets. The network communications link is used only when the packet is being transmitted.

PAD. (Packet Assembler Disassembler). The CTIX X.3/X.28/X.29 PAD is an application/presentation level installed service that allows any terminal to communicate with the X.25 Network Gateway.

PDN. Public Data Network.

pipes. "Pipes" are a programming feature of CTIX systems that allows terminal I/O to be redirected.

point-to-point. Point-to-point is a network configuration in which two points are connected to each other by a dedicated line, which can be a direct cable connection, a leased line, or a dialup to a service providing dedicated lines.

port. A port is the portion of a socket that specifies which logical input or output channel of a process is associated with the data.

process. A process is a program in execution. A source or destination of data from the point of view of the TCP or other host-to-host protocol.

protocol. A protocol, in general, is a set of rules that enable a network entity to understand a communicating entity; however, the entity that employs these rules, such as the transport level protocol, TCP, is commonly referred to as a protocol. Therefore, a protocol is a software entity that implements a specific layer or function in a network architecture.

In using the programmatic interface, a protocol is the next higher level protocol identifier, an internet header field.

reap. To reap children is to an activity of the parent process which prevents child processes that have died from accumulating as zombies. A child process dies when it exits. See **zombie**.

RFC. Request For Comment.

root. Root is the login name of the super user. The super user is the user who has the widest form of machine privileges.

routing. Dynamic, or adaptive, routing is the ability to transfer data automatically to the destination node via alternative paths consisting of one or intermediate nodes. Routing includes the ability to ascertain available paths and to decide the best path, taking into account topology changes or node failures as they occur.

search path. A predefined path of directories that the shell follows when looking for a file you have specified.

server. A server is a system service, called a demon. It is usually a user program that runs in background, in user space, to provide a defined set of functions to the user who uses it through the command interface. Each time a user invokes it, the server provides a separate process for that user.

shell script. A shell script program is a file containing a series of command language statements that are executed by the shell (<u>sh</u>(1)) to perform various functions, especially those using interprocess communication.

socket. A socket is a file descriptor made up of system of data structures and pointers used by the kernel to identify and keep track of a process. It is an address which specifically includes a port identifier, that is, the concatenation of an Internet Address with a TCP port. Sockets are transparent to the user. Programs must open sockets to access network functions. Any one process cannot have more than 2Ø open files at a given time.

super user. See **root.**

TCP. Transmission Control Protocol is a transport level, connection-oriented protocol that provides reliable end-to-end message transmission over an internetwork.

tuple. A tuple is a mathematical term for set of numbers composed of two or more factors. For example: [(XY)(AB)].

UDP. User datagram protocol is an unreliable user level transport protocol for transaction-oriented applications. It handles datagram sockets. It uses the IP for network services.

uname. Uname (UNIX name) is the name of the host machine. It is assigned in the file **/etc/rc.** The name must be in the file because the machine reads it at boot time to relearn its name.

user. The user of the internet protocol. This may be a higher level protocol module, an application program, or a gateway program. See also **Network User.**

VME. A standard microcomputer bus originated by Motorola. The MightyFrame provides for an optional upgrade which includes a bus translator board called the VME interface board. This board interfaces to the proprietary local I/O bus. The VME board accepts the Ethernet board.

volume home block. The volume home block is the disk block on MiniFrame and MightyFrame that describes the disk layout.

X.25. X.25 is a circuit-switched network protocol used commonly in Europe and less so in the United States. X.25 is based on a three-layer, peer-communications protocol standard defined by the International Telegraph and Telephone Consultative Committee (CCITT).

X.25 Network Gateway. The MightyFrame X.25 Network Gateway is a loadable server that provides in interface to an X.25 PDN for the CTIX TCP/IP-X.25 Interface and other applications.

X.25 Interface. See MegaFrame CTIX X.25 Interface or MightyFrame CTIX TCP/IP-X.25 Interface.

XNS. Xerox Networking System is a network and transport level protocol originally developed by Xerox to network Ethernet LANs. It is currently used widely to support Ethernet-like LANs and gateways.

zombie. A zombie is a child process that has died and not been reaped. See **reap.**