UNICOS® Networking Facilities Administrator’s Guide

S–2304–10011

New Features

UNICOS® Networking Facilities Administrator's Guide S–2304–10011

This rewrite of *UNICOS Networking Facilities Administrator's Guide* supports the 10.0.1.1 release of the UNICOS operating system. The following changes to this manual were made for the UNICOS 10.0.1.1 release:

- As a result of SPR 707731, the ICMS menu directions were fixed in the sections for "Configuring Network Interfaces" and "Creating the `hycf` File" in Chapter 2, page 3.
- Minor updates and corrections were made throughout this manual.

Record of Revision

Version	***Description***
9.0	August 1995 Original Printing.
10.0	October 1997 Update to support the UNICOS 10.0 release running on Cray computer systems.
10.0.0.1	January 1997 Update to support the UNICOS 10.0.0.1 release running on Cray computer systems.
10.0.0.2	May 1998 Update to support the UNICOS 10.0.0.2 release running on Cray computer systems.
10.0.0.3	October 1998 Update to support the UNICOS 10.0.0.3 release running on Cray computer systems.
002	July 1999 Update to support the UNICOS 10.0.0.6 release.
10010	October 2001 Update to support the UNICOS 10.0.1.0 release.
10011	May 2002 Update to support the UNICOS 10.0.1.1 release.

Contents

Page

Preface

This manual contains information on the administration of networking facilities supported by the UNICOS 10.0.1.1 operating system.

Warning: Starting with the UNICOS 10.0 release, the term *Cray ML-Safe* replaced the term *Trusted UNICOS*, which referred to the system configuration used to achieve the UNICOS 8.0.2 release evaluation. Because of changes to available software, hardware, and system configurations since the UNICOS 8.0.2 system release, the term *Cray ML-Safe* does not imply an evaluated product, but refers to the currently available system configuration that closely resembles that of the evaluated Trusted UNICOS 8.0.2 system.

For the UNICOS 10.0 release and later, the functionality of the Trusted UNICOS system has been retained, but the `CONFIG_TRUSTED` option, which enforces conformance to the strict B1 configuration, is no longer available.

UNICOS System Administration Publications

Information on the structure and operation of a UNICOS system, as well as information on administering various products that run under UNICOS, is contained in the following documents:

- *General UNICOS System Administration* contains information on performing basic administration tasks as well as information about system and security administration using the UNICOS multilevel security (MLS) feature. This publication contains chapters documenting file system planning, UNICOS startup and shutdown procedures, file system maintenance, basic administration tools, crash and dump analysis, the UNICOS MLS feature, and administration of online features.

- *UNICOS Resource Administration* contains information on the administration of various UNICOS features available to all UNICOS systems. This publication contains chapters documenting accounting, automatic incident reporting (AIR), the fair-share scheduler, file system quotas, file system monitoring, system activity and performance monitoring, and the Unified Resource Manager (URM).

- *UNICOS Configuration Administrator's Guide* provides information about the UNICOS kernel configuration files and the runtime configuration files and scripts.

- *UNICOS Networking Facilities Administrator's Guide* contains information on administration of networking facilities supported by the UNICOS operating system. This publication contains chapters documenting TCP/IP for the UNICOS operating system, the UNICOS network file system (NFS) feature, and the network information system (NIS) feature.
- *NQE Administration* describes how to configure, monitor, and control the Cray Network Queuing Environment (NQE) running on a UNIX system.
- *Kerberos Administrator's Guide* contains information on administration of the Kerberos feature, a set of programs and libraries that provide distributed authentication over an open network. This publication contains chapters documenting Kerberos implementation, configuration, and troubleshooting.
- *Tape Subsystem Administration* contains information on administration of UNICOS and UNICOS/mk tape subsystems. This publication contains chapters documenting tape subsystem administration commands, tape configuration, administration issues, and tape troubleshooting.

Related Publications

The following man page manuals contain additional information that may be helpful.

- *UNICOS User Commands Reference Manual*
- *UNICOS System Calls Reference Manual*
- *UNICOS File Formats and Special Files Reference Manual*
- *UNICOS Administrator Commands Reference Manual*
- *UNICOS System Libraries Reference Manual*

The following publication is useful for establishing connectivity between the High Performance Parallel Interface (HIPPI) network of a Cray mainframe and any host that has a physical path to any of the network interfaces of the Cray L7R.

- *Cray L7R Release Overview and Software Installation Guide*, contains information on the Cray L7R release and details regarding software installation and configuration for the Cray L7R. This publication contains chapters documenting an overview of the release, purpose and function of the Cray L7R, system and network configuration requirements, software installation and configuration instructions, and troubleshooting.

Design specifications for the UNICOS multilevel security (MLS) feature are based on the trusted computer system evaluation criteria developed by the U. S. Department of Defense (DoD). If you require more information about multilevel security on UNICOS, you may find the following sources helpful:

- DoD Computer Security Center. *A Guide to Understanding Trusted Facility Management* (DoD NCSC-TG-015). Fort George G. Meade, Maryland: 1989.
- DoD Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria* (DoD 5200.28-STD). Fort George G. Meade, Maryland: 1985. (Also known as the *Orange book*.)
- DoD Computer Security Center. *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria* (DoD NCSC-TG-005-STD). Fort George G. Meade, Maryland: 1987. (Also known as the *Red book*.)
- DoD Computer Security Center. *Summary of Changes, Memorandum for the Record* (DoD 5200.28-STD). Fort George G. Meade, Maryland: 1986.
- DoD Computer Security Center. *Password Management Guidelines* (CSC-STD-002-85). Fort George G. Meade, Maryland: 1985.
- Wood, Patrick H. and Stephen G. Kochan. *UNIX System Security*. Hasbrouck Heights, N.J.: Hayden Book Company, 1985.

Ordering Documentation

To order software documentation, contact the Cray Software Distribution Center in any of the following ways:

E-mail:
`orderdsk@cray.com`

Web:
`http://www.cray.com/craydoc/`

Click on the `Cray Publications Order Form` link.

Telephone (inside U.S., Canada):
1–800–284–2729 (BUG CRAY), then 605–9100

Telephone (outside U.S., Canada):
Contact your Cray representative, or call +1–651–605–9100

Fax:
+1–651–605–9001

Mail:
Software Distribution Center
Cray Inc.
1340 Mendota Heights Road
Mendota Heights, MN 55120–1128
USA

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items, such as file names, pathnames, man page names, command names, and programming language elements.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names. The following list describes the identifiers:

1	User commands
1B	User commands ported from BSD
2	System calls
3	Library routines, macros, and opdefs
4	Devices (special files)
4P	Protocols
5	File formats
7	Miscellaneous topics
8	Administrator commands

Some internal routines (for example, the `_assign_asgcmd_info`() routine) do not have man pages associated with them.

variable	Italic typeface indicates an element that you will replace with a specific value. For instance, you may replace *filename* with the name `datafile` in

	your program. It also denotes a word or concept being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.
[ ]	Brackets enclose optional portions of a syntax representation for a command, library routine, system call, and so on.
`...`	Ellipses indicate that a preceding element can be repeated.

The following machine naming conventions may be used throughout this document:

Term	Definition
Cray PVP systems	All configurations of Cray parallel vector processing (PVP) systems, including Cray SV1 series systems.
Cray MPP systems	All configurations of the Cray T3E series. The UNICOS operating system is not supported on Cray T3E systems. Cray T3E systems run the UNICOS/mk operating system.

Reader Comments

Contact us with any comments that will help us to improve the accuracy and usability of this document. Be sure to include the title and number of the document with your comments. We value your comments and will respond to them promptly. Contact us in any of the following ways:

E-mail:
`swpubs@cray.com`

Telephone (inside U.S., Canada):
1–800–950–2729 (Cray Customer Support Center)

Telephone (outside U.S., Canada):
Contact your Cray representative, or call +1–715–726–4993 (Cray Customer Support Center)

Mail:
Software Publications
Cray Inc.
1340 Mendota Heights Road
Mendota Heights, MN 55120–1128
USA

Introduction [1]

This guide is a teaching and reference document for people who manage the operation of Cray systems running the UNICOS operating system. It contains information on administration of networking facilities supported by the UNICOS operating system.

This manual provides information on the use and administration of the following network administration products:

- Transmission Control Protocol / Internet Protocol (TCP / IP), a networking protocol suite that allows Cray systems to be added to any established TCP / IP network and provides a large number of networks and users with access to Cray systems.
- Network file system (NFS); UNICOS NFS is a software product that allows users to share directories and files across a network of machines.
- Network information system (NIS) (formerly called yellow pages); UNICOS NIS is a network service that allows information such as passwords and group IDs for an entire network to be held in a single database.

Also included in this guide is an appendix listing supported management information base (MIB) variables.

This guide replaces neither experience nor other documents that more fully describe specific system areas. Familiarity with the reference publications listed in the preface, combined with the full-time efforts of an individual, is necessary to effectively manage Cray systems running the UNICOS operating system.

TCP/IP [2]

Transmission Control Protocol / Internet Protocol (TCP / IP) is an integrated networking protocol suite developed by the U. S. Department of Defense (DoD) for the Defense Data Network (DDN). The TCP / IP implementation described in this section is for the UNICOS operating system, running on all Cray systems. This implementation is compatible with other network communications products that comply with the DoD TCP / IP specifications. It allows Cray systems to be added to any established TCP / IP network and provides many networks and users with access to Cray systems.

The following hardware and software are minimum requirements for installing and operating TCP / IP:

- A Cray system running the UNICOS operating system.
- Network media, such as a Cray FEI-3 front-end interface, a Network Systems Corporation (NSC) Data Exchange (DX) or Data Exchange Extended Chassis (DXE) adapter, a Fiber Distributed Data Interface (FDDI), or a Cray High Performance Parallel Interface (HIPPI).

  Note: The required network media differs based on the type of Cray system you have.
- At least one other host in the network that supports DoD standard TCP / IP protocols.

This section provides you with the information you need to install and maintain TCP / IP. It is assumed that you have a working knowledge of your particular Cray system and the UNICOS operating system.

The following topics are discussed:

- TCP / IP basics
  - Details of Internet addressing, including hardware address resolution and subnet addressing
  - Name and address mapping process
  - Routing of data through gateways to its final destination
- Configuration for GigaRing, Model E (IOS-E), and Model V (IOS-V) based systems

  - Physical network media through which Cray systems interface with other systems
  - Directions for configuring TCP/IP (with or without the use of the UNICOS installation and configuration menu system (ICMS)). For more information on the use of the UNICOS ICMS, see *UNICOS System Configuration Using ICMS*.

- Network tuning
- Troubleshooting
- Trace facility
- Security administration basics

2.1 TCP/IP Basics

Cray systems operate in environments in which communication with computer systems in other networks is necessary. Communication among these computers is possible through a technology that connects the diverse systems and allows each to access the other. This technology, known as *internetworking*, The standards that specify the details of how these computers communicate, called Transmission Control Protocol/Internet Protocol (TCP/IP), were developed by the U. S. Department of Defense (DoD) Advanced Research Project Agency. This collection of individual networks in government, military, university, and industry installations is called the *TCP/IP Internet*, or simply the *Internet* provides gateways through which the data can travel.

This section presents the aspects of TCP/IP that you must know to administer the Cray system on the Internet. The following aspects are described:

- Internet structure
- Network hardware
- Internet addressing
- Hardware addressing
- Routing of information

2.1.1 Internet Structure

A computer network consists of two or more computers (also known as *hosts*) that are connected to each other by network technology such as an NSC HYPERchannel, a Cray FEI-3 front-end interface, an FDDI, or a Cray High Performance Parallel Interface (HIPPI). This type of connection is known as a *direct connection* and creates a network known as a *local area network (LAN)*. An *internetwork* consists of two or more interconnected LANs, which can be interconnected by either a host or a router. Hosts and routers that interconnect LANs are known as *gateways*.

Internet technology allows communication between networks (for example, a gateway receives a packet of data from a computer on one network and sends it to a computer on another network). Figure 1 shows gateway G connected to both networks A and B. Gateway G receives packets of data from any host on network A (dotted lines represent hosts) and transfers them to network B. Likewise, gateway G receives packets of data from any host on network B and transfers them to network A.

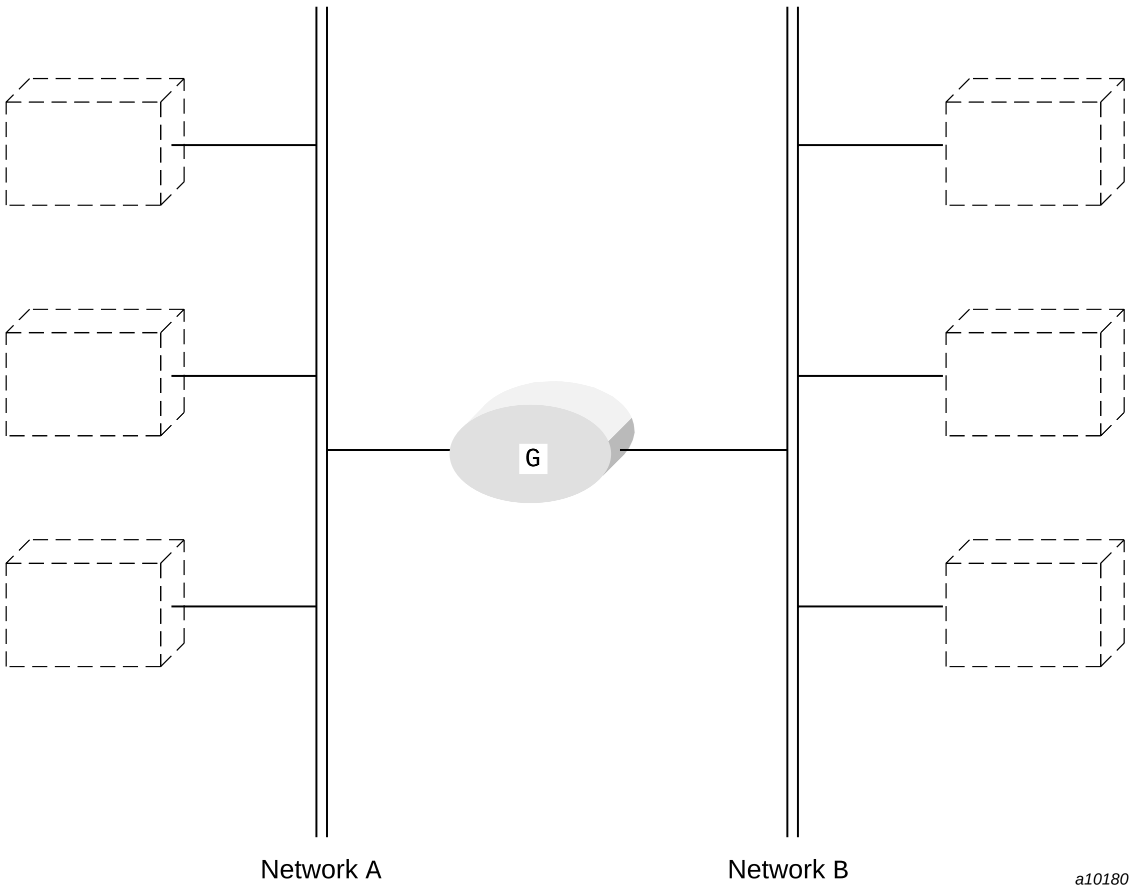


Figure 1. Two networks interconnected by gateway G

Typical internetworks are more complex than the one depicted in Figure 1. Packets are sometimes routed through numerous gateways before arriving at the designated destination. Sometimes decisions have to be made as to the most efficient route to take. Consider the internetwork in Figure 2. Dotted lines represent hosts.

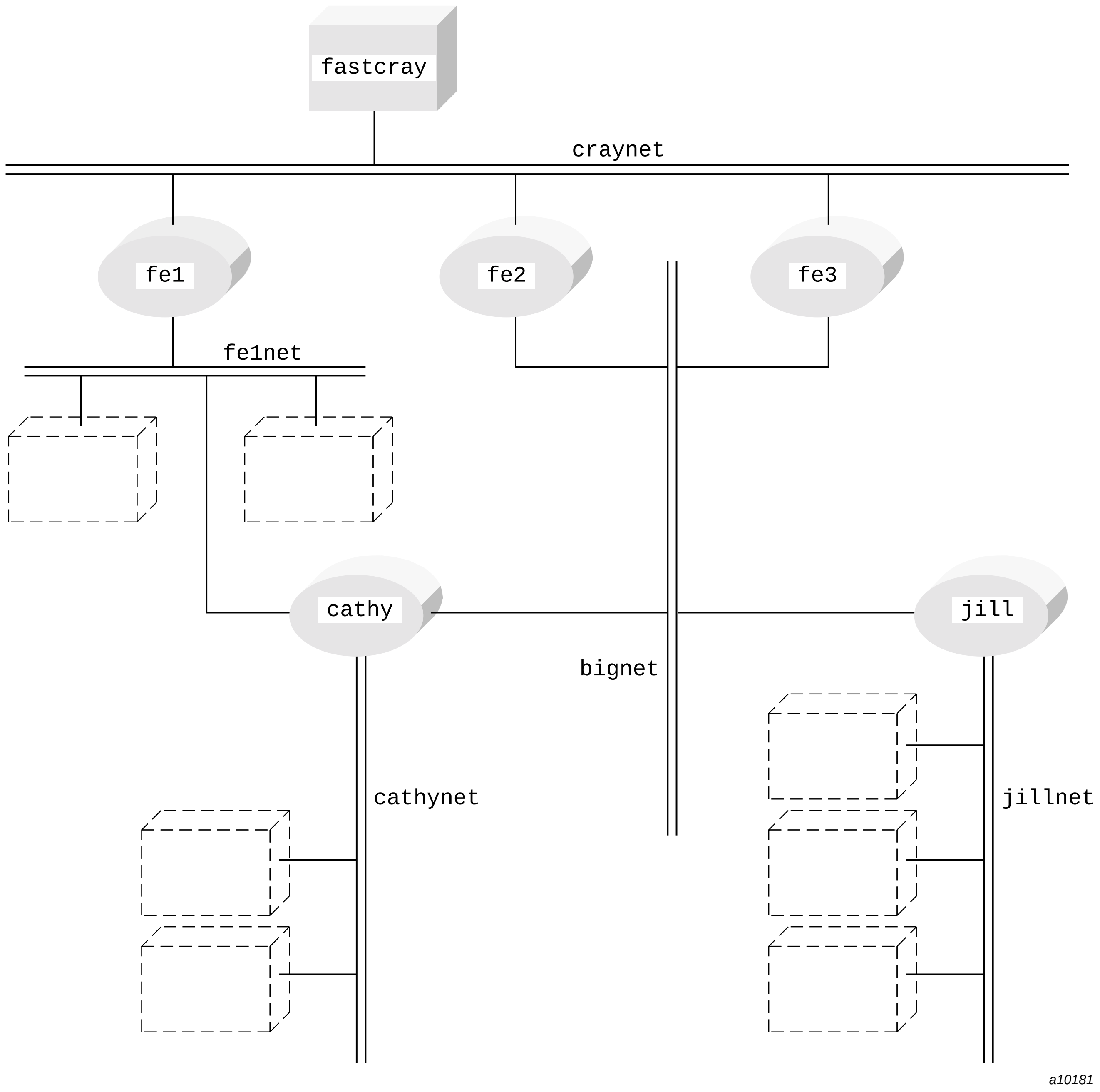


Figure 2. Several interconnected networks

Notice, for example, the choices for routes that a host on the `jillnet` network would have when trying to send data to the `craynet` network. Data could be routed through `fe2` or `fe3` to `craynet`, or through `cathy`, to the `fe1nt` network, and then through `fe1` to `craynet`. Planning the route that data will take to get from one network to another is the job of the administrator. See Section 2.1.4, page 14 for information on how this process works. See Section 2.3.3, page 148 for a discussion of how route selection can affect network performance.

2.1.2 Internet Addressing

The technology that allows hosts to interface with Cray systems requires that two types of addresses be specified: Internet and hardware. This section discusses Internet addressing; the following section discusses hardware addressing.

TCP/IP provides an addressing system in which each host on the internetwork is assigned at least one 32-bit address. This address is used for communication among the hosts in the internetwork and is known as the *Internet address*. Each host has as many Internet addresses as it has network interfaces. Those hosts with more than one network interface can be used as gateways. For example, the gateway called `cathy` in Figure 2, page 7 has three different Internet addresses because it interfaces with `cathynet`, `bignet`, and `fe1net`. Gateways `fe1`, `fe2`, `fe3`, and `jill` each have two Internet addresses. The hosts shown inside the networks have only one Internet address each because they are connected to only one network.

The Internet address consists of two main parts: the part that identifies the network to which the host is connected, and the unique part that identifies the host.

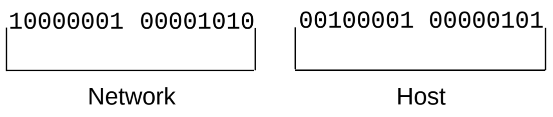
2.1.2.1 Address Classes

TCP/IP provides four types of addresses to accommodate the diversity of Internet structures. Class A addresses are assigned for networks that consist of more than 65,534 hosts. The high-order 8 bits identify the network; the remaining 24 bits identify the host. The first bit of a class A address is always `0`. The following Internet address is a class A address:

$$\underbrace{\texttt{00000001}}_{\text{Network}}\quad\underbrace{\texttt{00001010 00100001 00000101}}_{\text{Host}}$$

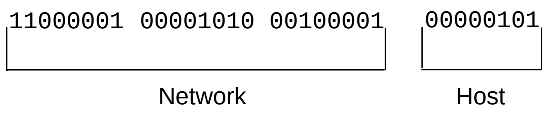
a10253

Class B addresses are assigned to internetworks that consist of from 254 to 65,534 hosts. The high-order 16 bits identify the network; the remaining 16 bits identify the host. The first and second bits of a class B address are always `10`. The following Internet address is a class B address:



a10254

Class C addresses are assigned to internetworks that consist of fewer than 254 hosts. The high-order 24 bits identify the network; the remaining 8 bits identify the host. The first three bits of a class C address are always `110`. The following Internet address is a class C address:



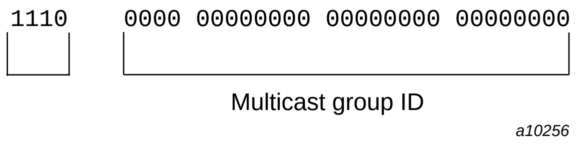
a10255

Table 1, page 9 summarizes the characteristics of class A, B, and C addresses.

Table 1. Characteristics of class A, B, and C addresses

Address class	Maximum hosts/network	Maximum unique networks
A	16,777,214	126
B	65,534	16,382
C	254	2,097,150

Class D addresses are assigned to IP multicast group addresses. A multicast address is used when sending an IP datagram to a group of hosts that belong to a specific multicast group. A multicast group address is the combination of the high-order four bits `1110` and the multicast group ID (the remaining 28 bits). The following Internet address is a class D address:



When written in decimal notation, these addresses range from 224.0.0.0 through 239.255.255.255. Some multicast group addresses are assigned as well-known addresses by the Internet Assigned Number Authority (IANA). These are called permanent host groups. For example, the address for the permanent host group that includes all routers on a specified subnet is 224.0.0.2. See RFC 1700 for more information about permanent host groups.

2.1.2.2 Subnet and Supernet Addressing

Some sites have numerous networks. If each network were assigned a different network identifier, the routing process would become difficult. *Subnet addressing* allows numerous networks to appear to the routing process to be part of the same network. This standard is defined by RFC 950. See Section 2.1.4, page 14, for details of the routing process.

Subnet addressing uses part of the host portion of the Internet address to designate subnetworks. Consider the class B address in the previous example. Ordinarily, the last 16 bits specify the host identifier. With subnetting, you can take part of the host portion to identify subnetworks and use the remaining part of the host portion to identify hosts on the subnetworks. For example, you could use the first 3 bits of the host portion to identify subnetworks. Using only 3 bits, you can identify as many as six different networks (`001`, `010`, `011`, and so on). Bits `000` and `111` are reserved. The other 13 bits could then be used to identify the hosts on the various subnetworks. You could identify as many as 8190 hosts per subnetwork this way.

The bits of the host identifier part of an Internet address can be divided in whatever manner is necessary to identify your particular network structure. For example, if you need more subnetworks, you can use more bits to identify the subnetworks and fewer to identify the hosts.

If it is necessary to choose subnet addressing for your site, you must also choose a 32-bit subnet mask to identify your network. This mask shows which part of an Internet address identifies the network (including the subnetwork) and which identifies a host. The mask must contain 1's to represent the bits that identify the network portion and 0's to represent the bits that identify the host portion. For example, suppose your site has been assigned the following class C address:

```
11000000 00001001 00101000 00000000
```

Assume that you need to choose a subnet mask to identify the network and host portions of this address. You know that the left three octets of the mask will be all 1's, signifying that the first three octets are the network portion (class C addresses always take 24 bits to identify the network portion). Because you are going to use part of the host portion (rightmost octet) to identify the subnetwork, some of the bits will be 1's, and some will be 0's in the last octet of the mask. How much of this octet you use to identify the subnetwork and how much you use to identify the hosts depends on the structure of your network. For example, if you needed to identify six physical subnetworks, you would set the first 3 bits of this octet to 1's. You would set the rest of the bits to 0's, to identify them as host bits. Those 5 bits could identify up to 30 hosts on each of the subnetworks. Your subnet mask would be as follows:

```
11111111 11111111 11111111 11100000
```

Maybe you have only two subnetworks to be identified. In that case, your subnet mask would be as follows:

```
11111111 11111111 11111111 11000000
```

With this subnet mask, you could identify up to 62 hosts on each network.

You must specify this mask in the `netmask` argument of the `ifconfig`(8) command. An example of using the `netmask` argument is shown on Section 2.2.9.6.2, page 112.

As the Internet continues to grow, the phenomenon of supernetting is starting to become more common. Supernetting is the reverse of subnetting. In it, a block of adjacent network addresses lying on a power-of-two boundary are combined into a single, larger network. For example, 16 class C network addresses are shown below:

```
11000000 00001001 00101000 00000000
11000000 00001001 00101000 00000000
11000000 00001001 00101000 00000000
11000000 00001001 00101000 00000000
```

These could be combined into a supernet for a site which needed an address space somewhere between those of a class B and a class C network.

For more information on supernetting, see RFC 1519.

2.1.2.3 Decimal Notation

For ease of use, Internet addresses are usually expressed as four decimal integers, separated by decimal points. Each decimal integer represents one octet of the 32-bit address. For example, the Internet address

```
10000001 00001010 00100001 00000101
```

would be expressed as follows in decimal notation:

```
129.10.33.5
```

A class A address (which always begins with a high-order binary 0) is recognized as any Internet address whose decimal notation begins with an integer in the range 0 to 127. A class B address (which always begins with a high-order binary 10) is recognized as any Internet address whose decimal notation begins with an integer in the range 128 to 191. A class C address (which always begins with a high-order binary 110) is recognized as any Internet address whose decimal notation begins with the integer in the range of 192 through 223.

Subnet masks can also be expressed as 4 decimal-separated numbers. The subnet masks in the previous examples would be expressed as 255.255.255.224 and 255.255.255.192, respectively.

2.1.2.4 Mapping Internet Addresses to Names

Because it is easier to remember a name than an Internet address, Internet protocol has provided a means by which you can assign names to all Internet hosts. To do so, create the `/etc/hosts` file or use the domain name service and create the `/etc/resolv.conf` file and enter the Internet addresses and corresponding names or aliases. See Section 2.2.4.1, page 31, or Section 2.2.8.3, page 77, for more details on the `/etc/hosts` file and the domain name service, respectively). Subsequently, when a user issues a command and uses a name instead of an Internet address, the appropriate database is searched, and the name is matched with the corresponding Internet address.

2.1.3 Hardware Addressing

Network hardware and the associated device drivers are protocol-independent. Therefore, the hardware cannot use Internet addresses to get the data to its remote destination.

The device drivers write hardware addresses (not Internet addresses) into the hardware message header. TCP/IP uses Internet addresses, however, to obtain hardware addresses. This section describes three methods for determining the destination's hardware address, given its Internet address. This process is called *hardware address resolution*.

2.1.3.1 Using the Address Resolution Protocol (ARP)

Address resolution protocol (ARP) is a dynamic address resolution protocol. Broadcast LAN media, such as Ethernet and FDDI, support ARP. ARP allows a host to find the physical address of a target host on the same physical network, given only the target host's Internet address. When a host that is connected to a broadcast type network wants to resolve an Internet address such as 128.150.30.10, it broadcasts a special packet (ARP request) that asks the host with Internet address 128.150.30.10 to respond with its physical (hardware) address (for example, 00-00-95-40-00-30). All hosts on the LAN receive the request, but only the host that recognizes its Internet address sends a reply that contains its physical address. ARP is used on Cray FDDI and Ethernet interfaces to implement the address resolution needed to map the 32-bit Internet address (logical address) to a 48-bit FDDI/ Ethernet hardware address (physical address).

ARP is a caching protocol; to avoid repeated use of the protocol, it maintains a cache of recently learned Internet-to-physical address mappings. For more information about ARP, see RFC 826.

2.1.3.2 Using a Configuration File—Model E and Model V Systems Only

For each type of Cray network hardware used in the internetwork, you can create a configuration file, in which you supply specific information about the hardware. In this file, you include Internet addresses and corresponding hardware addresses. (If you have created the `/etc/hosts` file, you can use names instead of Internet addresses.)

For Model E and Model V based systems, this configuration file is commonly referred to as the `/etc/hycf.` *name* file, or just the `hycf` file.

The `hycf` file is used as input to the `hyroute`(8) command, which does the actual mapping of the Internet addresses to the network hardware addresses (see Section 2.2.6.4, page 44, for details on the `hycf` file and the `hyroute` command). The `hycf` file information supplied to the `hyroute` command is the information that TCP/IP uses to build the message header (including the physical destination address and the logical destination address). Use this method if you want to control or customize any information in the message header. For HYPERchannel connections, because of the information that must be specified, you must use this method.

2.1.3.3 Using a Configuration File—GigaRing Based Systems Only

For each type of Cray network hardware used in the internetwork, you can create a configuration file, in which you supply specific information about the hardware. In this file, you include Internet addresses and corresponding hardware addresses. (If you have created the `/etc/hosts` file, you can use names instead of Internet addresses.)

The HIPPI `arp` file has the following format: `/etc/ghippi`*X*`.arp`

where *X* is the ordinal of the HIPPI interface. The `arp` file is used by `/etc/initif,` which performs `arp -s`.

The GigaRing `arp` file has the following format: `/etc/grX.arp`

where *X* is the ordinal of the GigaRing interface.

To use `arp` on a HIPPI or GigaRing interface, enter permanent ARP entries in the ARP table with the `arp -s` command. ATM interface configuration is done in the `atm.pvc` file. It is not necessary to configure an `arp` entry for interface names on FDDI or Ethernet.

2.1.3.4 Using the Internet Address—Model E and Model V Based Systems Only

If you do not create a `hycf` file, TCP/IP constructs the hardware address, as well as other control information, from the third and fourth octets of the Internet address, as follows:

- The third octet is interpreted as the physical destination (byte 4 of the message header).
- The fourth octet is interpreted as the logical destination (byte 5 of the message header).

For example, if the Internet address were 128.123.10.8, the message header `physical destination address` field would be 10; the message header `logical destination address` field would be 8.

2.1.4 Routing of Information

routing is the process of finding a path or route for data to travel from source to destination. This section describes *static routing*, in which the administrator plans the most efficient route for the data. *dynamic routing* is an automatic process in which the routes are chosen by use of the `gated`(8) command. See Section 2.2.8.1, page 61, for a description of dynamic routing.

If the source address and the destination address are a part of the same network, the route is considered *direct*. If the source and the destination are on different networks, gateways must be used between them. These routes are considered *indirect*. In some indirect Internet configurations, the path from source to destination is obvious, as shown in Figure 3.

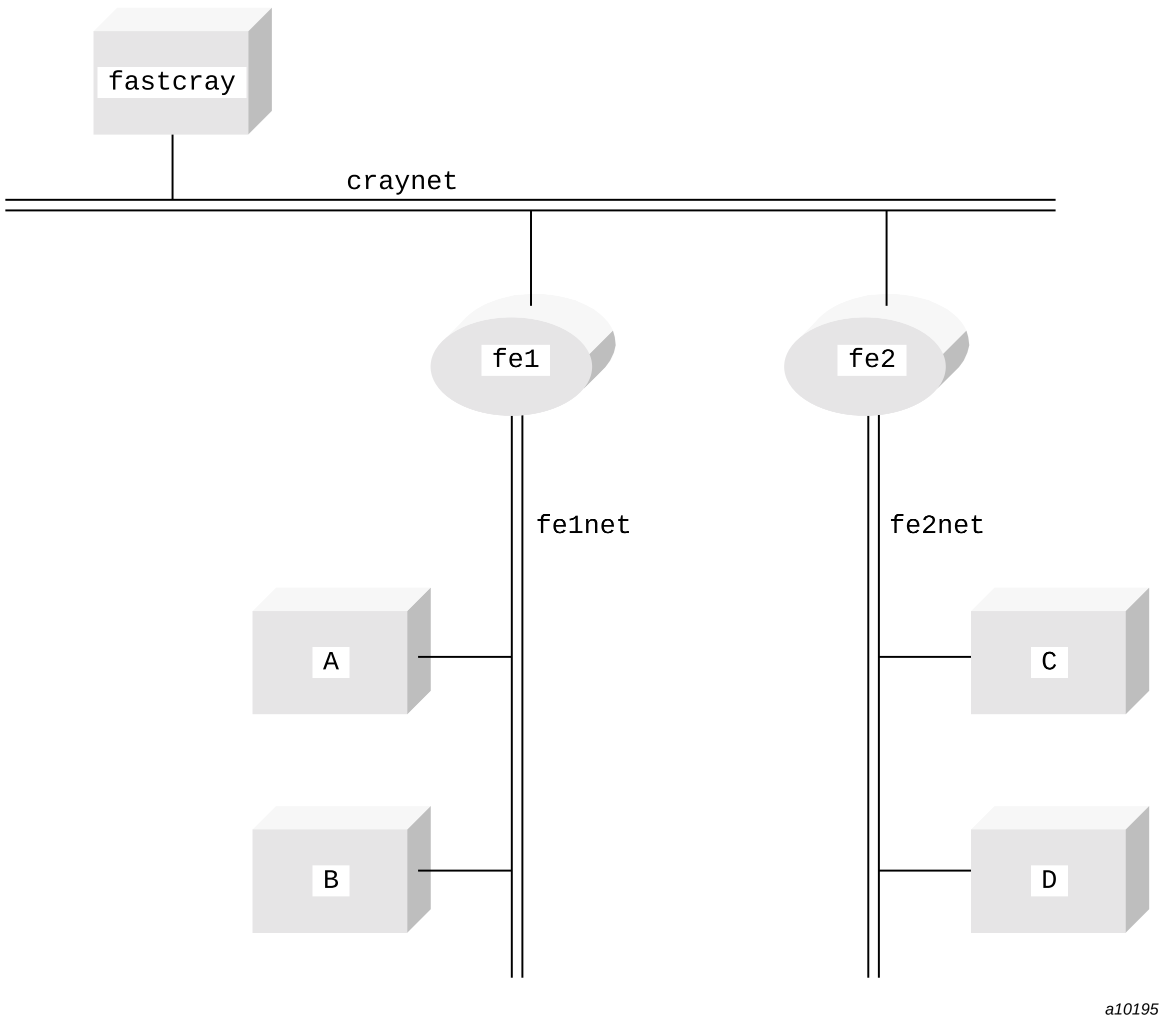


Figure 3. Internet configuration with obvious paths

If data is to be sent between network `craynet` and host `A` or host `B`, it must be routed through gateway `fe1`. If data is to be sent between `craynet` and host `C` or host `D`, it must be routed through gateway `fe2`. However, in Figure 4, the paths that the data must take to reach network `craynet` are not so obvious. Any of the hosts in `bignet` can have data routed through gateway `fe1` or `fe2`.

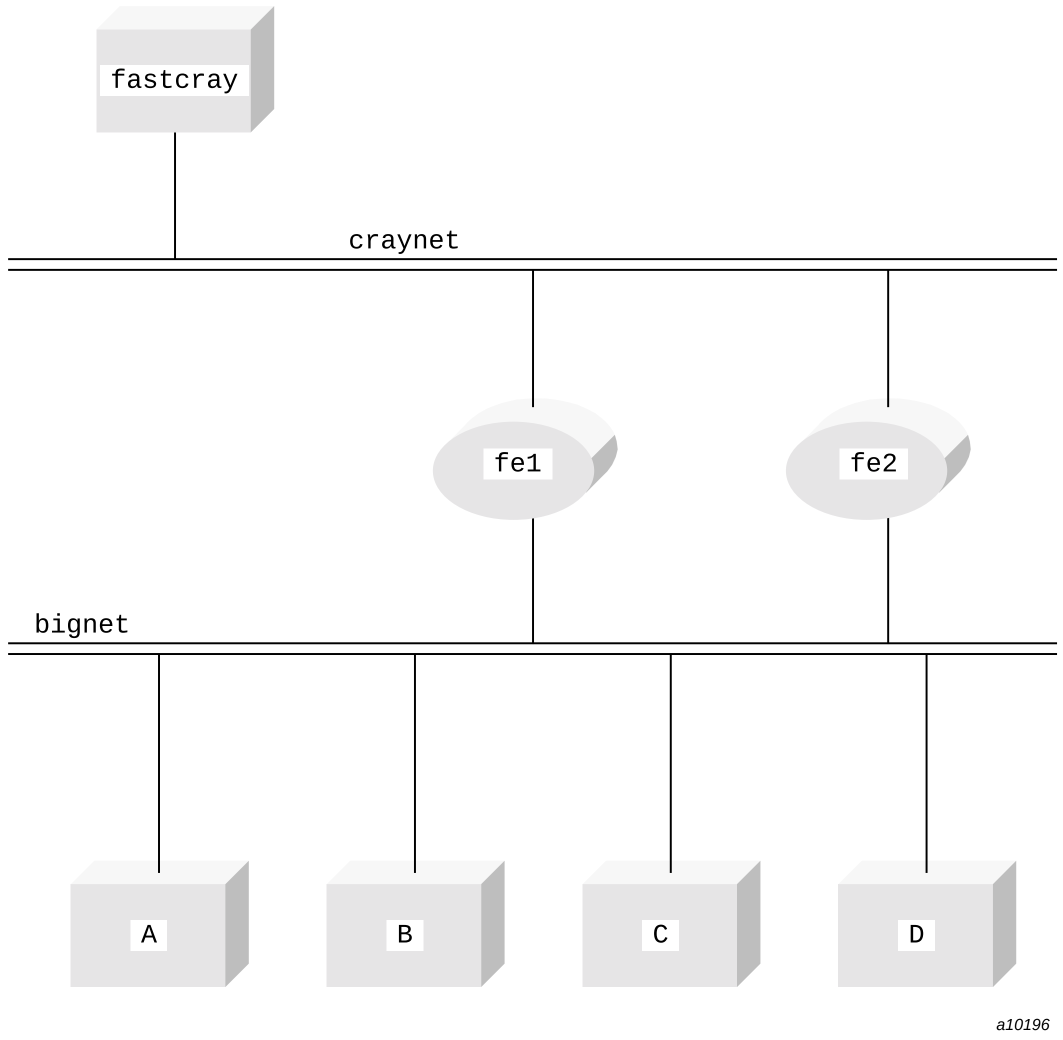


Figure 4. Internet configuration with alternative paths

2.1.4.1 Routing Procedure

As an administrator, you must determine the proper routing information, based on the configuration of your networks, for each host that is not directly connected to the Cray system. For example, if you were administering the internetwork shown in Figure 4, you would first determine which hosts will use which gateways to communicate with the `craynet` network. You might decide that hosts `A` and `B` will use gateway `fe1`, and hosts `C` and `D` will use gateway `fe2`. The routing information you would supply for each host would indicate the first gateway through which the data must travel to reach `craynet`. Each pass through a gateway is known as a *hop*.

If there is more than one local network to which gateways are connected, you must also specify a network route for each gateway. You can also specify default gateways to provide paths for the hosts or networks that have no alternative gateways to use.

Data is sent through the internetwork in pieces known as *datagrams*. When a host is ready to send a datagram to another host, the destination Internet address is included in the datagram, and the adapter encapsulates the datagram into a unit of data known as a *network packet*. A network packet can travel only between directly connected hosts.

In Figure 5, a datagram is routed from host `A` in network `bignet` to the `craynet` network.

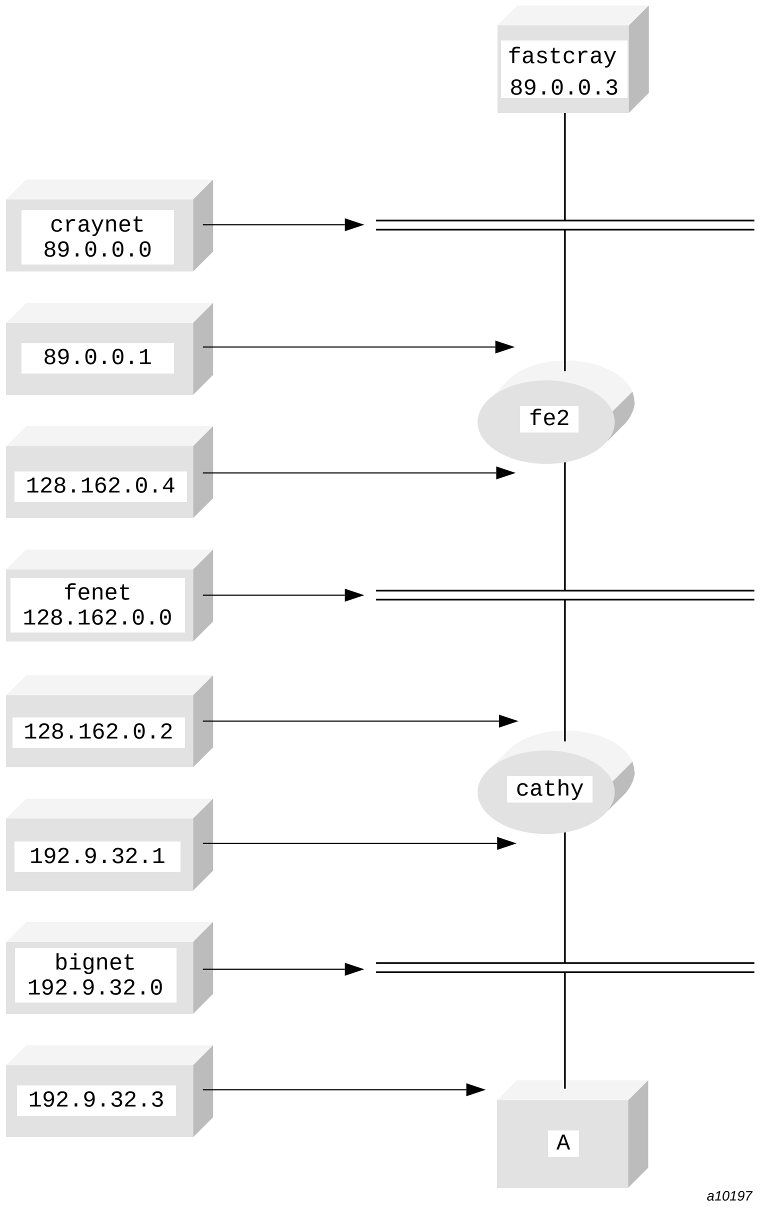


Figure 5. Routing procedure example

When host `A` (address 192.9.32.3) is ready to send a datagram to the `craynet` network (network address 89.0.0.0), the routing information for host `A` reveals that, for datagrams going to network 89.0.0.0, the first hop is gateway `cathy` (address 192.9.32.1). Although the ultimate destination is for a host on `craynet`, a packet is built and sent to `cathy` 's address. This is as far as this packet can go. When the packet reaches address 192.9.32.1, the datagram is stripped from the packet. The destination address in the datagram is checked against `cathy's` routing information. `cathy` 's information reveals that, for datagrams going to network 89.0.0.0, the first hop is gateway `fe2` (address 128.162.0.4). The datagram is put into a new packet, with the destination address specified as `fe2` 's address. When the packet reaches address 128.162.0.4, the datagram is again stripped from the packet. The routing information for `fe1` reveals a direct route (no hops) to network 89.0.0.0. The datagram is put into yet another packet, and the packet is sent to the destination host.

2.1.4.2 Routing Algorithm

You specify routes by creating kernel data structures known as *routing tables*. Routing tables contain *destination/gateway* pairs; *destination* is the destination of the datagram, and *gateway* is the gateway to be used for the next hop. A gateway can be either a host or an adapter. The example in the previous section is based on a routing algorithm that the routing software uses to find a match between a datagram's destination address and an address found in the destination field of the routing table. When the match is found, the corresponding gateway is used as the destination for the packet.

The algorithm is as follows:

1. If the destination address of the datagram exactly matches any of the destination addresses in the host-specific routes of the routing table, return the first such match.
2. Otherwise, if the network portion of the destination address of the datagram matches any of the destination network addresses in the network routes in the routing table, return the first such match.
3. Otherwise, if a default route is available, return that route.
4. Otherwise, there is not enough information to route the packet, and an error is returned to the sender. If this occurs, user programs receive either a `Host unreachable` or a `Network unreachable` error message.

2.1.4.3 Routing Tables

A route for each network interface connected to a host is the only information that the routing software needs to route data through the network. This information is contained in the routing tables. Routes for directly connected networks are set up automatically when an interface is initialized (see Section 2.2, page 22). However, the system administrator must set up routing tables on all hosts that will be communicating with host networks that are not directly attached. Each host has a start-up script, which is used to configure network interfaces and initialize routing information whenever the system is initialized. On many systems, the administrator places `route`(8) commands in the start-up script to place the proper routing information in the routing tables when the start-up script is executed.

The `/etc/staticrts` start-up script supplied with the UNICOS system takes a slightly different approach, requiring the system administrator to place the proper routing information in the `/etc/gated.conf` file, from which it will be extracted and used to initialize the routing tables when the `/etc/staticrts` script is executed.

Assume that you are the network administrator for the internetwork segment shown in Figure 6. Routing tables must be set up for hosts `A`, `B`, `C`, `D`, `E`, and `F`, and between `smallnet` and `bignet`. It is your job to determine the routing of data from these hosts to the Cray system (`fastcray`). (See Section 2.3, page 123, for details on selecting the most efficient routes.) The `bignet` network has four hosts, and network `smallnet` has only two. If all data coming from `bignet` and going to network `craynet` is routed through gateway `G2`, gateway `G2` might be too busy, and routing might become unnecessarily slow. Therefore, to minimize the load on gateway `G2`, you can decide to route data from hosts `A`, `B`, and `D` through gateway `G2`, and to route data from host `C` through gateway `G1`.

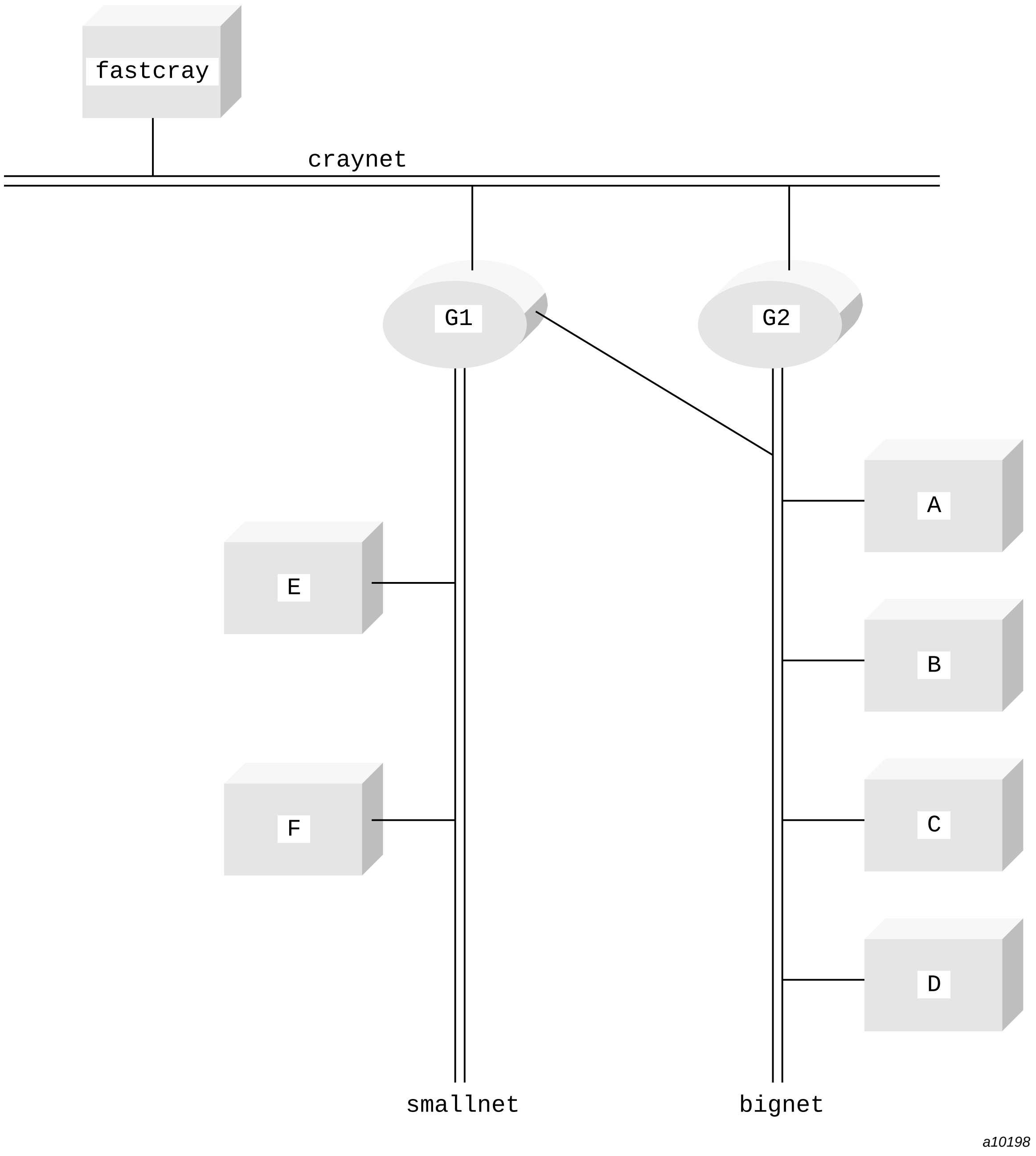


Figure 6. Routing table example

The following list shows the routing tables set up from the `route` commands for each host on the internetwork shown in Figure 6, page 21:

Host	Routing table

`A`	`route add craynet G2`
	`route add smallnet G1`

`B`	`route add craynet G2`
	`route add smallnet G1`

`C`	`route add default G1`
`D`	`route add craynet G2` `route add smallnet G1`
`E`	`route add default G1`
`F`	`route add default G1`

`fastcray`	`route add smallnet G1`
	`route add bignet G2`
	`route add C G1`

The routing tables set up for hosts `A`, `B`, and `D` specify that gateway `G2` will be used for network `craynet` destinations, and gateway `G1` will be used for `smallnet` destinations. The routing table for host `C` specifies that, no matter what the destination, data from host `C` will be routed to gateway `G1`. The routing tables for hosts `E` and `F` also indicate that gateway `G1` will be used for all routing. Hosts `E` and `F` have no other choice. The routing table for `fastcray` is set up so that the route it uses to send data to each host is the same route that the host uses to send data to `fastcray`.

2.2 Configuring TCP/IP

This section describes the configuration of TCP / IP in a UNICOS environment. It describes the system components that are configurable, and it discusses methods for configuring them. The following steps are described:

- Configuring the TCP kernel code (determining and setting mbufs, and configuring other kernel parameters by using the `netvar`(8) and `sysctl`(2) commands)
- Setting up network-wide configuration files (`/etc/hosts` and `/etc/networks`)

- Setting up local system configuration files (`/etc/services`, `/etc/shells`, `/etc/hosts.equiv`, `/etc/config/spnet.conf`, `/etc/protocols`, and `/etc/config/interfaces`)
- Configuring network interfaces and daemons
- Performing start-up procedures
- Using the `telnet linemode` feature
- Assisting users in setting up environments (tab settings, `$HOME/.netrc`, `$HOME/.rhosts`, and `bftp`)

If you are planning to run the UNICOS security feature on your Cray system, refer to Section 2.6.3, page 216, for additional kernel configuration.

If you are also planning to tune your system, refer to Section 2.3, page 123.

2.2.1 Configuration Issues

This section describes several instances in which the configuration you choose will affect the general operation of your TCP/IP networking software. Some of these considerations involve choices among different methods by which the software can accomplish something (such as looking up host name and address information); others involve general policy considerations affecting the configuration of your network (such as the selection of appropriate Internet addresses); and, if used, the proper network configuration for installation and configuration of the Cray L7R.

2.2.1.1 Looking up Host Names and Addresses

The underlying TCP/IP protocol suite identifies and communicates with all hosts by their numeric Internet addresses. If the software that implements the protocols allowed users to specify hosts by only numeric Internet addresses, however, keeping track of which numeric address referred to which host would become a tremendous burden on users (not to mention system administrators). To avoid this problem and to make networking easier to use, the TCP/IP software allows for central lookup of host names for conversion into the numeric Internet addresses used by the underlying protocols. (It is easier to remember names than to remember numbers.) UNICOS TCP/IP provides two separate configuration methods to allow the software to map between Internet addresses and host names. The method you choose for your system configuration has a large impact on the networking capabilities provided to your system’s users.

> **Note:** These host address look-up methods are contained in library routines common to all TCP/IP networking software; no software needs recompilation, regardless of the configuration you choose.

The first method of mapping between host addresses and names is to place the host addresses and names in the static text file `/etc/hosts`, as described in Section 2.2.4.1, page 31. The advantage to using the `/etc/hosts` file is simplicity; the information your users are seeking resides in one, easy-to-find and easy-to-fix file. The disadvantage is that the `/etc/hosts` file on your Cray system is merely a local copy of information that must be consistent across all hosts on your networks; it is very easy for the information in your `/etc/hosts` file to become out-of-date because of changes that occur to remote networks at your site (for example, the addition of a new host, or a changed name of a host).

The second method is to use the Internet domain name service, the configuration of which is described in Section 2.2.8.3, page 77. The domain name service is a distributed collection of name servers on various hosts; these servers answer requests for information about mappings between host addresses and names. That is, the UNICOS TCP/IP software requests that a server (either on your local Cray system or a remote system, according to your configuration) supply the address for a given host name, and the server responds with the information (possibly after consulting other name servers).

> **Note:** The use of the domain name service is not allowed for the Cray ML-Safe configuration of the UNICOS system.

The disadvantages of using the domain name service are that using this service is more complex than relying on one `/etc/hosts` file (and, therefore, somewhat more complex to administer) and that relying on name servers configured by other administrators for information about those administrators' remote hosts and networks introduces the risk of incorporating mistakes from other administrators' information.

The advantages of using the domain name service are that, despite the complexity of the domain name service, relying on it for information about changes to remote hosts and networks is a lot less work than repeatedly updating the `/etc/hosts` file by hand; also, your Cray system users can access more remote hosts more conveniently and without administrator intervention.

However, the use of the domain name service for your host name and address lookup does not completely eliminate the need for an `/etc/hosts` file. Because the domain name service uses the underlying TCP/IP protocols to ask information of and receive information from other name servers, system startup can be a problem if the domain name service is requested to resolve a host name that is necessary for network initialization (for example, the name of a local interface to be configured up) before the networks have, in fact, been initialized. Thus, even when using the domain name service, you must have in your `/etc/hosts` file the names of any hosts referred to during your start-up procedure. Typically, this list includes, at a minimum, all the hosts in the network access list (NAL) and workstation access list (WAL), the names of any local interfaces, and the names of any remote systems for which you will be configuring routing information.

In summary, relying on `/etc/hosts` for your Cray system should be adequate for sites with smaller networks administered by one central authority. However, if your Cray system will be attached to the Internet or some other larger collection of networks that are supported by many different administrators, the use of the domain name service will provide more reliably consistent host name and address mapping than will the use of an `/etc/hosts` file alone.

2.2.1.2 Selection of Internet Addresses

If you are connecting your Cray system to an existing TCP/IP network, Internet addresses have already been established for the networks at your site; also, there is likely to be an established list of host names and addresses for configuration of your `/etc/hosts` file, or one or more existing domain name servers on various hosts on the network to resolve host names and address for your system.

If, however, you are establishing a new TCP/IP network to which to attach your Cray system, you must use a new network address. The UNICOS TCP/IP software places no specific policy restrictions on the network numbers you use for the addresses on any specific networks at your site; technically, you can choose any number you want for a new network address. However, choosing a network number in such a haphazard manner introduces the risk of conflicting network numbers if your site decides to connect its TCP/IP networks to those of another site.

To avoid conflicts of this type, administration centers exists for official Internet network addresses. See The Accredited Registrar Directory at the following Internet address for a listing of accredited agencies:

`http://www.internic.net/regist.html`

For example, to request official network addresses for any networks at your site, you could send your request to VeriSign, Inc. (formerly Network Solutions, Inc.) at the following location:

VeriSign, Inc.

505 Huntmar Park Drive

Herndon, VA 20170

U.S. & Canada: 1-877-699-3243

Worldwide: +1 703-742-0914

E-mail Address: http://www.netsol.com/cgi-bin/help/contactus

Attn: Global Registry Services

See RFC 1400 for background information.

(In practice, sites with several local networks will typically request one official network address for the site and differentiate among local networks by establishing a different subnet address for each one, not by giving each local network its own official network address.)

Note: Because it is often very difficult to foresee the future needs of a given networking installation, it is strongly recommended that you secure official Internet addresses from the InterNic for any TCP/IP networks at your site, even if you have no current plans to connect your site to any outside networks. This small precaution will prevent the trouble of converting your network's Internet addresses if your site should decide to connect to the Internet in the future.

2.2.1.3 Using a Cray L7R

As with any Internet Protocol (IP) router, the Cray L7R provides connectivity between various networks. The Cray L7R is intended to act as a specialized router (gateway) between the High Performance Parallel Interface (HIPPI) network of a Cray mainframe and any host that has a physical path to any of the network interfaces of the Cray L7R, for example, a Gigabit Ethernet network.

Note: To obtain a current list of supported networking hardware, contact your Cray service representative.

The Cray L7R specialized function is to off-load much of the Cray mainframe resources spent dealing with IP networking traffic. This improves the TCP bandwidth per connection between the Cray mainframe and host systems.

For information about installing and configuring the Cray L7R in your network, see the *Cray L7R Release Overview and Software Installation Guide*.

2.2.2 Configuring the TCP/IP Kernel Code

This section describes the steps you must take to determine and set segments of a special memory pool (*mbufs*), and to specify other kernel parameters through the use of the `netvar`(8) facility. See Section 2.3.2.1, page 129, for a more detailed description of mbufs.

2.2.2.1 Determining the Number of Mbufs Needed

To determine the number of mbufs to allocate, you must consider the following factors:

- Whether HIPPI is being used as a TCP/IP interface. HIPPI does not achieve its peak performance with TCP/IP unless you use large kernel buffers (`TCP_SNDBUF` and `TCP_RCVBUF` socket options) and expanded windows (`TCP_WINSHIFT` socket option). UNICOS `ftp` and `ftpd` facilities automatically use these to maximize file transfer throughput.

  Note: For Model E and Model V based systems, define a minimum of 1800 mbufs for each HIPPI interface configured for TCP/IP; for GigaRing based systems, define a minimum of 4000 mbufs for each HIPPI interface configured for TCP/IP.

- Mbufs are incremented by 4000. See Section 2.2.2.2.1, page 28 for a table showing the number of mbufs required to `ifconfig` each interface type.

- Number of routing entries in the routing table (for Model E and Model V based systems, each entry requires 1 mbuf; for GigaRing based systems, each entry requires 2 mbufs.). See Section 2.1.4, page 14, for a description of routing tables.

- Number of active sockets (for Model E and Model V based systems, each requires 3 mbufs for the life of the connection; for GigaRing based systems, each requires 2 mbufs for the life of the connection). See Section 2.3.2.1.4, page 135, for a description of sockets.

- Number of UNICOS network file system (NFS) user ID maps configured (approximately 150 mbufs for each).

- On systems running the UNICOS security feature, the number of NAL, WAL, and Internet Protocol Security Options (IPSO) map entries (each requires 1 mbuf).

These factors account for a small, but fixed pool of mbufs. Additional mbufs are required to handle transient peaks, packet headers, data queued for input or output to active connections, and file transfer and remote shell connections.

To avoid a setting that is too low, you should set your number of mbufs to 50% above the minimum estimate. For more information on selecting the number of mbufs to allocate, see Section 2.3.2.1.3, page 131.

2.2.2.2 Setting the Number of Mbufs

To set the number of mbufs, perform the following steps:

1. If you are using the UNICOS ICMS to configure your kernel, consult the `Configure System -> Kernel configuration` menu for the item that sets the appropriate number of mbufs.

   If you are not using the UNICOS ICMS to configure your kernel, set (or change) the value of the `TCP_NMBSPACE` parameter in the `/usr/src/uts/cf/config.h` file to the appropriate number of mbufs.

   Either method sets the number of mbufs, which will be allocated at system initialization time for TCP/IP use. You cannot change the size of this pool while the system is running.

2. Issue the `nmake`(1) command from the `/usr/src/uts/cf` directory (for Cray J90 systems, the `/usr/src/uts` directory). This recompiles the module with the new mbuf value and links a new kernel.

3. Issue the following command from the `/usr/src/uts/cf` or `/usr/src/uts` directory:

   ```
   nmake installsys
   ```

   This installs the newly built kernel with the new number of mbufs in the root directory.

2.2.2.2.1 Mbuf Requirements—for GigaRing Based Systems Only

1. The following shows the number of mbufs required to `ifconfig` each interface type:

Interface	mbufs required
ATM	2000
Ethernet	500
FDDI	1000
HIPPI	4000

`GigaRing`	`4000`

2.2.2.3 Specifying Other Kernel Variables

You can use either the `netvar`(8) or `sysctl`(8) command to set or change other kernel networking variables, such as the TCP / IP `send` and `receive` space, while the system is up and running. At system start-up time, the default `tcpstart`(8) script, which is supplied with the UNICOS system, uses `netvar`(8) to initialize kernel variables to values you select (see Section 2.2.9.2, page 107, for details).

2.2.3 Security Configuration Parameters for Networking

Security configuration parameters for the UNICOS operating system are set in the kernel parameters and `config.h` file. For information on installing and configuring the UNICOS system, see *UNICOS System Configuration Using ICMS*.

The following are the security configuration parameters for networking:

Parameter	Description
`NFS_SECURE_EXPORT_OK`	Flag indicating whether information on a secure file system can be exported over NFS. A *secure file system* refers to a file system that has been explicitly marked as secure by `labelit`(8) or `mkfs`(8). This configuration parameter is retained for compatibility with the UNICOS operating system. If `NFS_SECURE_EXPORT_OK` is enabled (set to 1), remote access to secure file systems will be allowed. If `NFS_SECURE_EXPORT_OK` is disabled (set to 0), remote access to secure file systems is prohibited. It is recommended that sites use a setting of 1 for all configurations of the UNICOS operating system. The default setting is 1. There is no required ML-Safe setting.
`NFS_REMOTE_RW_OK`	Flag indicating whether remotely mounted NFS file systems can be mounted as read / write or whether they must always be read-only. If `NFS_REMOTE_RW_OK` is enabled (set to 1), remotely mounted NFS file systems may be mounted read / write or read-only. If `NFS_REMOTE_RW_OK` is disabled (set to 0), all remotely mounted NFS file systems are mounted read-only.

The default setting is 1. There is no required ML-Safe setting.

`SECURE_NET_OPTIONS`

A bit mask of flags that may contain the following bits. It allows a site to configure certain global networking options.

Bit name	Value	Description
`NETW_STRICT_B1`	00001	Restrict network label ranges so that NAL entries must either specify an IP security option or a single label for a remote host.
`NETW_SOCK_COMPAT`	00002	Automatically make all sockets set up by a privileged process multilevel sockets.
`NETW_RCMD_COMPAT`	00004	Allow traditional processing of `.rhosts` and `hosts.equiv`.

The default setting is 6 (`NETW_SOCK_COMPAT | NETW_RCMD_COMPAT`).

The 1 (`NETW_STRICT_B1`) bit is required for the Cray ML-Safe configuration of the UNICOS system.

2.2.4 Setting up Network-Wide Configuration Files

This section describes the configuration and format of the `/etc/hosts` and `/etc/networks` files, which map Internet addresses to host and network addresses, respectively. Because the Internet addresses listed in these files must be consistent across all hosts on the affected networks, many sites maintain master copies of these files (maintained by a central administrator), and when necessary, distribute updated copies to all of the systems that are based on the UNIX system and attached to their networks. This administrative technique is much simpler than relying on the individual administrators of each host to make changes to the files manually; thus, it is particularly recommended for sites with many networked hosts. However, if you ever need to modify the contents of your `/etc/hosts` or `/etc/networks` file directly, you must adhere to the formats specified in the following section.

2.2.4.1 The `/etc/hosts` File

The `/etc/hosts` file is a text file that associates each Internet address with one or more host names. See Section 2.2.1.1, page 23, for information on setting up the `/etc/hosts` file. There should be one entry in the `/etc/hosts` file (that is, one Internet address) for each network interface on each host of the networks that are accessible to your Cray system. The `gethost`(3) library routines, which are compiled into the TCP/IP software, consult the entries in the `/etc/hosts` file to map a host name supplied by the user to an Internet address (and vice versa).

If you are using the UNICOS ICMS to configure your Cray system, you can disable the appropriate portion of the UNICOS ICMS and manually update the `/etc/hosts` file. This would be appropriate for a site at which the `/etc/hosts` file for all systems that are based on the UNIX system is maintained by a central administrator and distributed to your Cray system. To complete the disable procedure, consult the `Configure System -> Configurator automation options` menu for the correct entry to use to automate the host address configuration.

If you decide to use the UNICOS ICMS to configure your `/etc/hosts` file, the `Configure System -> Network configuration -> General network configuration -> Host address configuration` menu lets you specify the remote hosts that are accessible to your Cray system, and it generates an `/etc/hosts` file from the information you supply.

If you decide not to use the UNICOS ICMS to configure your `/etc/hosts` file, or if you are not using the UNICOS ICMS for your configuration at all, you must follow the guidelines in this section to edit the `/etc/hosts` file directly to specify the remote hosts that are accessible to your Cray system.

The following is an example of an `/etc/hosts` file:

```
# /etc/hosts
# TCP/IP hosts file
#
127.0.0.1     localhost   loopback
89.0.0.0x3    r-n-d       rocky
89.0.0.4      support     bullwinkle        bw
89.00.0.5     cray-a      peabody
192.9.0.5     cray-b      boris
35.01.0.11    star
51.0.0.1      accounts
```

Use the following format rules for creating the `/etc/hosts` file:

- Begin each line with an Internet address. Each segment of the address can be expressed in decimal, octal, or hexadecimal format. Octal segments are preceded by 0; hexadecimal segments are preceded by 0x. In the previous example, the fourth segment of the address for host `r-n-d` is in hexadecimal format; the second segment of the address for host `cray-a` is in octal format; the second segment of the address for host `star` is in octal format; all other segments are in decimal format. Each Internet address must appear on a separate line.
- Follow the Internet address by at least one blank space or tab.
- Look at the official host name in the field following the Internet address. Ensure that the host name does not exceed 63 characters and contains only alphanumeric characters or the minus sign with the first character alphanumeric.
- Allow the host name to be followed by blank space and one or more aliases. In the previous example, host `support`, which has Internet address 89.0.0.4, has the two aliases, `bullwinkle` and `bw`.
- Always use the Internet address for the loopback test (see "The `hit`(8) command" and Section 2.4.1.1.3, page 170, for details of the loopback test). In the previous example, the Internet address 127.0.0.1 for `localhost` is used for the loopback test.

  Note: In the Cray ML-Safe configuration of the UNICOS system, special steps must be taken to run the `hit`(8) command (see *General UNICOS System Administration*, for more details on non-TCP software and UNICOS security).

- Add comment lines to the file by preceding the comments with a #. You can enter comments on a separate line, with a # as the first character, or on part of a line. Any data following a # and contained on the same line as the # is considered a comment.
- If you insert multiple entries for a single remote host, the first entry is used.

For improved performance, you can also create `/etc/hosts.bin`, a binary equivalent of the `/etc/hosts` file that can be searched more rapidly. If this binary file is present, it is used instead of the `/etc/hosts` file to discover the Internet address for a given host name. Use the `mkbinhost`(8) command to create the `/etc/hosts.bin` file. If you are using the default `tcpstart` script, this file is created automatically when the system is initialized.

2.2.4.2 The /etc/networks File

The /etc/networks file is a text file that associates names with networks that are accessible to your Cray system by using the network portion of the Internet addresses of the hosts on each network. (This is referred to as the *network number* of the network.) You can choose not to provide any mapping between network numbers and network names (that is, you can decide not to provide an /etc/networks file). If you do not provide an /etc/networks file, however, all references to networks in your configuration (typically as an argument to the route(8) command as part of the NAL and WAL configurations, or in the output of the netstat(1B) command) must be by network number, and not by name. If you do provide an /etc/networks file, the getnet(3) library routines, which are compiled into the TCP/IP software, consult the entries in the /etc/networks file to map a network name that is supplied by the user to a network number (and vice versa).

If you are using the UNICOS ICMS to configure your Cray system, you can disable the appropriate portion of the UNICOS ICMS and manually update the /etc/networks file. This would be appropriate for a site at which the /etc/networks file for all systems that are based on the UNIX system is maintained by a central administrator and distributed to your Cray system. To complete the disable procedure, consult the Configure System -> Configurator automation options menu for the correct entry to use to automate the network address configuration.

If you decide to use the UNICOS ICMS to configure your /etc/networks file, the Configure System -> Network configuration -> General network configuration -> Network address configuration menu lets you specify the networks that are accessible to your Cray system, and it generates a /etc/networks file from the information you supply.

If you decide not to use the UNICOS ICMS to configure your /etc/networks file, or if you are not using the UNICOS ICMS for your configuration at all, you must follow the guidelines in this section to edit the /etc/networks file directly to specify the networks that are accessible to your Cray system.

The following is an example of an /etc/networks file:

```
# /etc/networks
# TCP/IP networks file
#
loopback   127          # Class A network
craynet1   128.162.1    # Class B subnetted network
craynet2   128.162.2    #
ether1     128.162.3    r-n-d     research
```

Use the following format conventions for creating or changing the network file:

- Use the first field of every line as the official network name. The network name must not exceed 63 characters and must contain only alphanumeric characters and the minus sign with the first character alphanumeric.

  Follow the network name by blank space (that is, by any number of spaces or tabs).

- Note that the second field contains the network portion of an Internet address, which can be in decimal, octal, or hexadecimal format. (Ensure that each network number appears on a separate line.)

- Follow the network number by a blank space and one or more aliases. In the previous example, network `ether1`, which has network number 128.162.3, has the two aliases `r-n-d` and `research`.

- Remember that the `/etc/networks` file is searched sequentially. If there are multiple entries for one network, the first entry is used.

- Use network number 127 for loopback processing. Because loop-back processing is used to test the network, you should never use the official network name as an alias for `loopback`.

- Add comment lines to the file by preceding the comments with a #. Enter comments on a separate line, with a # as the first character, or on part of a line. Any data following a # and contained on the same line as the # is considered a comment.

2.2.5 Setting up Local System Configuration Files

The configuration files that are described in the following section must be configured if your site uses the facility that the file represents.

2.2.5.1 The/`etc/services` File

The `/etc/services` file is a text file that associates the name of a service with the protocol and standard port number that are used by the daemon that provides the service. The default `/etc/services` file that is supplied with the UNICOS system lists the Internet-standard protocols and ports for many services. Therefore, you should never need to modify an existing entry. However, it might be appropriate to add information about new services that are local to your site.

If you are using the UNICOS ICMS to configure your Cray system, you can disable the appropriate portion of the UNICOS ICMS and manually update the `/etc/services` file. To do so, consult the `Configure System -> Configurator automation options` menu for the correct entry to use to automate the services configuration.

If you decide to use the UNICOS ICMS to configure your `/etc/services` file, the `Configure System -> Network configuration -> General network configuration -> Networking services configuration` menu lets you specify the services available on your Cray system, and it generates an `/etc/services` file from the information you supply.

If you decide not to use the UNICOS ICMS to configure your `/etc/services` file (or if you are not using the UNICOS ICMS at all), you can add services by editing the `/etc/services` file directly and then running the `rsvportbm`(8) command. This command prevents servers that use the `bindresvport`(3) and `rresvport`(3) library routines from using the ports reserved in the `/etc/services` file.

The `/etc/services` file contains the following information:

- Official service name
- Port number (see Assigned Numbers, RFC 1010)
- Slash (/)
- Protocol name (currently TCP, UDP, or OSI)
- Aliases of the service

The following is a sample network services file:

```
# /etc/services
# Network Services, Internet style
echo         7/tcp
echo         7/udp
netstat     15/tcp
ftp         21/tcp
telnet      23/tcp
smtp        25/tcp      mail
hostnames  101/tcp      hostname
sunrpc     111/tcp
sunrpc     111/udp
# Host specific functions
tftp        69/udp
finger      79/tcp
# UNIX specific services
```

```
ntp        123/udp
exec       512/tcp
login      513/tcp      log
shell      514/tcp      sh
ntalk      518/udp
printer    515/udp
nqs        607/tcp
```

2.2.5.2 The `/etc/shells` File

The `/etc/shells` file is a text file that contains a list of shells that are associated with user accounts. The `ftpd`(8) program on the Cray system consults the list of shells in this file (by using the `getusershell`(3) library routine). If the login shell of an account to which someone is trying to log in through `ftp`(1B) does not appear in the `/etc/shells` file, `ftpd`(8) does not permit the login. If this file does not exist on your Cray system, it is because only accounts that list the standard shell (`/bin/sh`) or the C shell (`/bin/csh`) as their login shell are permitted to access the Cray system by using `ftp`(1B) to transfer files.

If you are using the UNICOS ICMS to configure your TCP/IP software, consult the `Configure System -> Network configuration -> TCP/IP configuration -> Shells` menu to supply the list of shells for the `/etc/shells` file.

If you are not using the UNICOS ICMS to configure your TCP/IP software, or if you are not using the UNICOS ICMS at all, you must edit the `/etc/shells` file directly.

The following shows a sample file that enables accounts that list the standard, C, and Korn shells as their login shell to access files on the Cray system by using `ftp`(1B).

```
# /etc/shells
# List of acceptable shells for chsh/passwd -s
# Ftpd will not allow users who do not have one of these
#     shells to connect
#
/bin/sh
/bin/csh
/bin/ksh
```

2.2.5.3 The `/etc/hosts.equiv` File

The TCP/IP `/etc/hosts.equiv` file is an optional file that provides host access permission information. The presence of a remote host's name in this file grants access to users on that host who have the same account name on the local host without requiring them to enter a password.

Note: The implications of `/etc/hosts.equiv` are different when you are running UNICOS security. See Section 2.6, page 211, for more details.

Following is an example of a `hosts.equiv` file:

```
# hosts.equiv
twg                    # Allows access to users from twg who have
                       # accounts on the local host
cray1     - steve      # Allows all users from cray1 except steve
cray2     mark         # Allows user mark to access all accounts on the
                       # local host
```

Use the following format conventions for creating or changing the `/etc/hosts.equiv` file:

- Begin each remote host's name on a separate line.
- Follow the host name by a blank space and the login name of any user on the host. The specified user can access all user accounts except root on the local host. Use a minus sign (`-`) in the second field to deny access to specific users.
- Use the wildcard character (*) in either field to match any user or host name.

  Note: It is a security risk to use the wildcard character. For example, the wildcard character in the second field allows all users on the remote host to access all user accounts on the local Cray host. Do not use the wildcard character without a thorough understanding of its implications.

The `rlogind`(8), `rshd`(8), and `rexecd`(8) daemon processes use this file; therefore, listing a remote host in `/etc/hosts.equiv` allows users of that remote host to access the local host through the r-series commands.

Note: You can prohibit users on all remote hosts from using the r-series commands by commenting out the references to r-series daemons (`rshd`, `rlogind`, `rexecd`, and so on) in the `/etc/inetd.conf` file. See Section 2.2.8.7, page 98, for more information.

Users from remote hosts who have accounts on the local host are automatically logged in and given execution privileges if their login names are the same on both the remote host and the local host. (Users can set up `$HOME/.rhosts` files in their home directories on the local host to allow automatic login and access to their accounts, even if the login names are different on the Cray system and the remote host.)

When using the `rlogin`(1B) program or `rexec`(3) library routine, users are prompted for their login names, and possibly their passwords, when the contents of the `.rhosts` file requires it. When using the `rcp`(1) or `rsh` (see `remsh`(1B)) program, users are denied access if neither `/etc/hosts.equiv` nor `$HOME/.rhosts` is configured. See the *TCP/IP Network User's Guide* for the format of entries in the `.rhosts` file.

A user with the login name `root` on the remote host can log in as `root` on the Cray system only when listed in the Cray system `.rhosts` file. You must carefully set up the `/etc/hosts.equiv` and `/.rhosts` files to minimize security risks.

The r-series programs (`rlogin`(1B), `rsh`(1B), `rcp`(1B), and `rexec`(3)) work only between hosts that run under operating systems that are based on a UNIX operating system. These programs provide automatic user authentication and automatically pass terminal information to the remote host by using the following procedure:

1. If the user is not root, the server searches `/etc/hosts.equiv` for the remote host's name.
2. If the host name is not found, the server searches for the user's login name in `/etc/passwd`.
3. If the login name is found, the user's `.rhosts` file is checked.
4. If the remote host name and remote user name are contained in the `.rhosts` file, the user is immediately logged in to the local host.

Warning: Using the second field of the `/etc/hosts.equiv` file is a security risk if `NETW_RCMD_COMPAT` is also enabled. The first field is a remote system name; the second field is an optional user name. For any remote system (`R`) and any user (`U`) that are listed in the first and second fields of the `/etc/hosts.equiv` file, `U` can log in to any local account, except `root`, from `R`, without using a password.

When `NETW_RCMD_COMPAT` is used, automatic authentication is a security risk because passwords are not checked if a user's local login and local host name are listed in the remote `.rhosts` file. For example, a user with the login name `joan` on any remote host that is in `/etc/hosts.equiv` is allowed automatic login to local account `joan`.

Another network security problem involves the function of the second field in the `/etc/hosts.equiv` file (when `NETW_RCMD_COMPAT` is running). After a remote host name, you can enter the user name of any user on the remote host. Then, when the `-l` *username* option is entered with any of the r-series commands, the user in the `/etc/hosts.equiv` entry has automatic login and file access to the accounts of every user on the local host except `root`.

For example, suppose you enter the following line in the `/etc/hosts.equiv` file on the local host:

```
twghost  mark
```

User `mark` can then enter the following command from host `twghost`:

```
$  rlogin  runcray  -l  steve
```

User `steve` now has automatic login and file access to the accounts of every user on the local host except `root`.

2.2.5.4 The `/etc/protocols` File

The `/etc/protocols` file is a text file that associates the name of a protocol in the Internet protocol suite with the protocol number (and one or more aliases). The default `/etc/protocols` file that is supplied with the UNICOS system contains a standard list of protocols and, therefore, might not need to be modified. However, it might be appropriate to add information about new protocols that are local to your site.

If you are using the UNICOS ICMS to configure your TCP/IP software, you can disable the appropriate portion of the UNICOS ICMS and manually update and configure the `/etc/protocols` file. To do so, consult the `Configure System -> Configurator automation options` menu for the correct entry to use to automate the protocols configuration.

If you are using the UNICOS ICMS to configure your `/etc/protocols` configuration file, the `Configure System -> Network configuration -> TCP/IP configuration -> Protocols` menu lets you specify the protocols available on your Cray system, and it generates a proper `/etc/protocols` file from the information you supply.

If you are not using the UNICOS ICMS to configure your `/etc/protocols` file, or if you are not using the UNICOS ICMS at all, you can add protocols by editing the `/etc/protocols` file directly. For each protocol listed in the file, there should be one line containing the official protocol name, the protocol number, and any aliases for the protocol name. The items are separated by any number of blanks or tab characters (or both). A comment is identified by a #.

The following is an example `/etc/protocols` file:

```
# /etc/protocols

ip     0     IP     # internet protocol, pseudo protocol number
icmp   1     ICMP   # internet control message protocol
tcp    6     TCP    # transmission control protocol
udp    17    UDP    # user datagram protocol
```

2.2.6 Configuring Network Interfaces—Model E and Model V Systems Only

The following section describe the following steps that you must take to configure each Cray system interface on the network:

1. Define the hardware devices
2. Name the Cray interface
3. Choose an Internet address
4. Create the `hycf` file, if needed

Note: Before you begin the following steps, ensure that all of the hardware diagnostic tests have been performed and that all of the hardware is functioning properly. Consult the appropriate vendor documentation for specific diagnostic information.

If you are using the UNICOS ICMS for configuration, you can use the `Configure System -> Network configuration -> TCP/IPConfigure System -> Kernel Configuration -> Communication Channel Configuration` menu and then the appropriate submenu, `Low-speed Channel Configuration` or `High-speed Channel Configuration` to configure your interfaces.

2.2.6.1 Defining Hardware Devices

Defined hardware devices for Cray systems are as follows:

- Model E systems

  Hardware devices for Model E systems are defined in the `network` section of the `~cri/os/uts/param` file on the OWS-E or the `/etc/config/param` on the Cray mainframe. Following is a sample `param` file for a Cray Model E system:

```
network {
      .
      .
      .
       npdev   0 {
                iopath { cluster 0;eiop 0;channel 030;}
                np_spec FEI3;
        }
      .
      .
      .

        hidev   0 {
                iopath {cluster 0;eiop 3;channel 030;}
                logical path 0 {flags 00;I_field 00;ULP_id 00;}
                flags 00;
                input;
                device type PS_32;
}
```

- Cray J90 Model V systems

  Hardware devices for Cray J90 systems are defined in the `network` section of the `/sys/param` file. The `/sys/param` file is located on the console disk for Cray J90 systems. The HIPPI interface is not defined in this file because it is detected during the boot sequence. Following is a sample `param` file for a Cray Model V system:

```
network {
     ...
     endev   0 {
             iopath { cluster 1; eiop 0; channel 020; }
     }

     fddev   0 {
             iopath { cluster 2; eiop 0; channel 040; }
        }
     atmdev  0 {
             iopath { cluster 0; eiop 0; channel 020; }
        }
     atmdev  1 {
```

```
                iopath { cluster 3; eiop 0; channel 020; }
        }
        ...
}
```

2.2.6.2 Naming the Cray Interface

The interface name, which TCP/IP uses to access a given hardware device, consists of two parts. The first part, which is the interface name prefix, indicates the type of hardware. The second part, which is the number of the interface name (also known as the *interface number*), identifies the physical device.

2.2.6.2.1 Interface Name Prefix

The interface name prefix is derived from the UNICOS driver that controls the hardware device. This prefix indicates the major number that TCP/IP uses when it defines the character special device for the given hardware. Following is a list of the prefixes and a description of the associated interfaces:

Prefix	Type of interface
`en`	Ethernet interface on Model V systems. Shown as `endev` in the `param` file.
`fd`	FDDI interface on Model E systems. Shown as the `fddev` hardware device entry in the `/etc/config/param` file.
`fddi`	FDDI interface on Model V systems. Shown as `fddev` in the `param` file.
`hi`	HIPPI interface on Model E and Model V systems. Shown as the `hidev` hardware device entry in the `/etc/config/param` file (for Model E systems only).
`np`	Low-speed HYPERchannel interface on Model E systems. Shown as the `npdev` hardware device entry in the `/etc/config/param` file.
`atm`	Asynchronous Transfer Mode (ATM) interface on Model V systems.
`bbgx:atmx`	ATM interface on Model E systems.

The following matrix summarizes the prefixes that Cray systems support:

Table 2. Supported Prefixes on Cray Systems

Cray Systems	Supported Prefixes						
	`en`	`fd`	`fddi`	`hi`	`np`	`atm`	`bbgx:atm`
IOS Model E		x		x	x		x
IOS Model V	x		x	x		x	

2.2.6.2.2 Interface Numbers

Interface numbers for hardware devices are derived from the order in which the hardware device entries appear in the files in which they are defined.

On Model E systems, the interface number for hardware devices is derived from the order in which the hardware device entry appears in the `network` section of the `param` file. This number is used to distinguish interfaces of the same type.

The following example shows the `network` section of a `param` file:

```
network {
      .
      .
      .
       npdev   0 {iopath { cluster 0; eiop 0; channel 030;}
                np_spec FEI3;
        }

        npdev   1 {iopath { cluster 0; eiop 0; channel 032;}
                np_spec N130X;

        hidev   0 {iopath { cluster 0; eiop 3; channel 030;}
                logical path 0 {flags 00; I_field 00; ULP_id 00;}
                flags 00;
                input;
                device type PS_32;
}

        hidev   1 {
                iopath {cluster 0; eiop 3; channel 032;}
                logical path 0 {flags 00; I_field 00; ULP_id 00;}
                flags 00;
                output;
                device type PS_32;
}
```

The interface names for this example are as follows:

```
np0
np1
hi0
hi1
```

2.2.6.3 Choosing an Internet Address

Each interface on your system must have an Internet address. The network and subnet parts of the Internet address of the interface must correspond to the Internet address of the network to which the interface is attached. The host portion of the address can be selected by using the preferred administrative practices for your site. (Some sites have a central network administrator who determines the new Internet addresses for new systems and interfaces; other sites let the individual system administrator decide.)

When you are choosing an Internet address for an interface that is attached to a new TCP/IP network, see Section 2.2.4, page 30, for guidelines.

2.2.6.4 Creating the `hycf` File

Note: For software loopback interfaces, direct FDDI interfaces, Ethernet interfaces, and ATM interfaces, it is not necessary to create an `hycf` file.

If you are using the UNICOS ICMS for configuration, you can use the `Configure System -> Network Configuration -> General network configuration -> Network hardware address configuration` menu to create and maintain the `hycf.`*name* file. The information you put in this file is used as input to the `hyroute`(8) command, which initializes the interface with hardware address information. This configuration file, sometimes known as the `hycf` file, varies according to the type of network hardware used, as shown in the following sections.

2.2.6.4.1 NSC Low-speed Connections

Note: Cray J90 systems support NSC boxes if you connect to a HYPERchannel network A400, N400, or DX adapter.

Use the following format when you create the `hycf` file for NSC low-speed connections:

connection hostname hardware_address `ff00 0 [`*mtu*`];`

connection	Type of connection. The `direct` parameter indicates that the host is connected to the same HYPERchannel trunk as the Cray system. The `gateway` parameter indicates that the host resides on a remote HYPERchannel trunk that is connected to the local trunk. **Note:** If the connection is type `gateway`, the fields that follow *hostname* must contain the names of hosts that are connected to the bridge. These hosts must be defined as direct hosts in this `hycf` file and must precede the gateway definition.
hostname	Host name that appears in the `/etc/hosts` file, or Internet address that is expressed in decimal notation.
hardware_address	Host's 16-bit HYPERchannel address in hexadecimal format. The most significant octet (byte) is the physical unit address that is configured within the network adapter hardware or firmware. Consult the documentation for your adapter for the best way to retrieve this information. The least significant octet is the logical address. This corresponds to the low-level device driver's logical path. By convention, logical path 5 is used for TCP/IP. However, the system administrator can choose any value. The only restriction is that the logical address chosen must be unique with respect to all protocols that use the device. The install tool does not support an alternate logical path for TCP/IP. To use a logical path other than 5, you must manually create the TCP/IP nodes. To do so, use the following command:

`mknod` *name* `c 35` *(npdev* `*16)` + *logical path*

The *npdev* variable indicates the low-speed device ordinal. The parameter *name* should be `/dev/comm/tcp` *xxxx* where *xxxx* is the four-digit octal representation of the minor device number. For example, to create a TCP/IP node for logical path 3 on device ordinal 1, you would use the following command:

```
mknod /dev/comm/tcp0023 c 35 19
```

The hardware address is included in the packet that is sent on the network to the destination host. The adapter that is attached to the destination host recognizes that the packet should be passed to the destination host by the physical unit address part of the hardware address. When the destination host processes the packet, the destination protocol is chosen based on the logical address. That is, the physical unit address tells which adapter box on the HYPERchannel should process the packet, and the logical address tells which protocol in the destination machine will process the incoming packet.

For example, suppose that you are setting the *hardware_address* field for a Cray system that has an adapter with physical unit address 0x52. Also assume that TCP/IP is configured to open logical path 5 for that device. The *hardware_address* field for that computer is 5205.

`ff00`	HYPERchannel control number in hexadecimal format (trunks to try, control bits).
`0`	Access code. This number must be 0.
mtu	Maximum transmission unit. The largest amount of data that can be sent to the host in one packet. The default is 4144. See Section 2.3.1, page 123 for details on specifying *mtu*.

The following is an example of an `hycf` file for NSC low-speed connections. Note that some of the connections do not use logical path 5 for TCP/IP.

```
# Configuration table for the hyroute command for NSC low-speed
# connections
#
#direct          MACH-NAME   TO                        MTU
direct           cray2       c605     ff00     0       4144;
direct           cray1       c205     ff00     0       4144;
direct           wilbur      12c1     ff00     0       4144;
direct           orville     2200     ff00     0       4144;
direct           wk01        2605     ff00     0       4144;
direct           wk02        4623     ff00     0       4144;
direct           wk03        4400     ff00     0       4144;
direct           wk04        4402     ff00     0       4144;
```

2.2.6.4.2 FEI-3 or VAXBI Connections

Note: Cray J90 systems do not support FEI-3 connections.

Use the following format when creating the `hycf` configuration file for FEI-3 or VAXBI connections:

`direct` *hostname dest* `ff00 0 [`*mtu*`];`

`direct`	Type of connection. `direct` indicates a direct connection between the Cray system and the host.
hostname	Host name that appears in the `/etc/hosts` file, or the Internet address expressed in decimal notation.
dest	Hardware address of FEI-3 or VAXBI in hexadecimal format. The leftmost 2 digits of the address must match the last field of the device's entry in the `comm_info` structure. The hardware address can be any number that is unique among the entries in the `comm_info` structure. This structure is found in the `/usr/src/uts/cf/conf.` *sn* `.c` file; *sn* refers to the serial number of a Cray system. The rightmost 2 digits are the logical address, and they are chosen by the system administrator. The only restriction is that they must be unique with respect to all protocols that use the device.
`ff00`	This field is unused by the Cray system.
`0`	This field is unused by the Cray system.
mtu	Maximum transmission unit. The largest amount of data that can be sent to the host in one packet. The default is 4144. See Section 2.3.1, page 123, for details on specifying *mtu*. For FEI-3 connections, *mtu* should be set to 4352. See *fy Driver Administrator's Guide.*

The following is an example of an `hycf` file for FEI-3 and VAXBI connections.

```
# Configuration table for the hyroute command for FEI-3/VAXBI connections

#direct    MACH-NAME     TO      NA      NA    MTU
direct     snq1-vme      1e02    ff00    0     4144;
direct     yafs-vme      1e03    ff00    0     4144;
```

2.2.6.4.3 HIPPI Connections

Use the following format when you create the `hycf` file for the HIPPI connections:

`direct` *hostname ifield readdev writedev* [*mtu*];

`direct`	Type of connection. `direct` indicates a direct connection between the Cray system and the host.
hostname	Host name that appears in the `/etc/hosts` file, or Internet address expressed in decimal notation.
ifield	HIPPI I-field value for connection to this host (hexadecimal notation). The I-field is typically used to make a HIPPI connection through one or more HIPPI crossbar switches. The I-field is used by the switch(es) to determine the path that is required to complete the connection. In the most simple case, the I-field contains the camp-on bit plus the port number to which the host is connected. The camp-on bit is bit 2**24 (0x01000000). This bit directs the switch to keep trying to make the connection until the connection is completed or the source abandons the connection attempt. The Cray HIPPI TCP/IP implementation is designed to be used with the camp-on bit if a switch that supports this feature is present. Some switches support additional routing implementations in which the destination address is not a simple port number but is instead a value that the switch converts into one or more port numbers. Consult the switch manufacturer's documentation for details on constructing an appropriate I-field. The I-field is significant even when no switch is present. If two systems are directly connected, you must still specify an I-field for both systems: the system you are currently configuring and the system to which you are connected. Each I-field must be unique. If they are equal, TCP/IP traffic

is not sent over the physical interface and all operations fail. The camp-on bit is not significant for connections in which no switch is present.

If a double-wide (64-bit) HIPPI connection can be made between two systems, the I-field must include bit 2**28 (0x10000000) for the driver to make use of this capability. If this bit is not set in the I-field, data is sent in 32-bit mode even though the 64-bit capability exists. Only the IOS–E can be configured to support the 64-bit HIPPI interface. Some switches and HIPPI fiber optic channel extenders offer 64-bit capability as well. If a switch offers both 64-bit and 32-bit capabilities, the `hycf` file should use the double-wide bit for hosts to which a 64-bit connection can be made. If a host has double-wide capability, but another host is only 32-bit capable, the entry for this host should not have the double-wide bit set. The driver ensures that the connection is made in the appropriate mode on a per-packet basis. The double-wide bit is also significant in the case in which no switch is present.

readdev	Minor device number (in hexadecimal format) of the character special device that TCP / IP will open for reading. See Section 2.2.6.4.4, page 50, for more information.
writedev	Minor device number (in hexadecimal format) of the character special device that TCP / IP will open for writing. See Section 2.2.6.4.4, page 50, for more information.
mtu	Maximum transmission unit. The largest amount of data to send to the host in one packet. The default is 16496. See Section 2.3.1, page 123, for details on specifying *mtu*.

The following is an example of an `hycf` file for HIPPI connections:

```
#
#                        NSC PS32 HIPPI Switch
#
#                                   Minor number
#        Hostname        I-field      In     Out      MTU
#      -----------      --------    ----    ----    -----
direct sn1001-hippi     03000000    0060    0070    4352  ; # Port 0
direct sn1701-hippi     03000001    0020    0030    4352  ; # Port 1
direct sn1601-hippi     03000002    0020    0030    4352  ; # Port 2
direct uss-hippi        03000003    0020    0030    4352  ; # Port 3
direct sn1061-hippi     03000004    0020    0030    4352  ; # Port 4
```

2.2.6.4.4 Selecting `readdev` and `writedev` Values

The read and write minor device numbers, `readdev` and `writedev`, determine the logical path that will be opened and provide the mapping to physical HIPPI channels for this interface. A range of 16 minor numbers maps to each physical HIPPI channel. Minor numbers 0 through 15 map to the first channel, numbers 16 through 31 map to the second and so on. The path number is the remainder of the minor number divided by 16. For example, minor number 0 is path 0, as is minor number 16. Minor number 1 is path 1 on the first HIPPI channel. Minor number 17 is path 1 on the second HIPPI channel (17 mod 16 equals 1).

Path 0 is the dedicated path; other paths are shared paths. The dedicated path can be opened only if no shared paths are currently open. Opening the dedicated path prevents the subsequent opening of shared paths on that channel. TCP/IP requires only one path and can be brought up on either the dedicated path or on a shared path. If TCP/IP is brought up on a shared path, user programs with appropriate privilege can share the HIPPI channel by opening character special device files. It should be noted that user programs can interfere with TCP/IP traffic, causing performance degradation. You should ensure the proper functioning of applications that you allow to share the HIPPI channel with TCP/IP. Among other things, these applications must adhere to the HIPPI-FP standard for HIPPI packet formation. They must use an upper-level protocol identifier (ULP-ID) other than 4; TCP/IP uses 4. If TCP/IP is brought up on a shared path, performance is slightly lower than if it is brought up on a dedicated path, even if no user program is currently active on another path.

On Model E systems and Cray J90 systems, there is also an I/O control-only path. On this path, the `open`(2), `close`(2), and `ioctl`(2) system calls can be issued, but attempts to transfer data are rejected. Usually, this path is used for gathering statistical data from the device driver. The `xnetmon`(8) utility is an example of this type of program.

The I/O control-only path is defined as `MAXPATHS-1`. On Model E systems, `MAXPATHS` is set to the *himaxpaths* variable in the `param` file. For example, if *himaxpaths* is set to 8, path 7 becomes the I/O control-only path. On Cray J90 systems, the `param` file is not used to configure HIPPI channels. `MAXPATHS` is always set to 16, which means that path 15 is always the I/O control-only path.

The methods of determining which physical channel is assigned to each minor number range are different for Model E systems and memory HIPPI systems.

- Model E systems

  The method for associating a minor device number range for a physical HIPPI channel on Model E systems includes making an entry in the network section of the `param` file for each physical HIPPI channel. These entries begin with `hidev`. Entry 0 (`hidev 0`) is the first HIPPI channel, `hidev 1` is the second, and so on. The first device is associated with minor numbers 0 through 15, the second with 16 through 31. Following is an example of the HIPPI portion of the network section of a Model E `param` file:

```
hidev   0 {
        iopath {cluster 0; eiop 3; channel 030; }
        logical path 0 {flags 00; I_field 00; ULP_id 00;}
        flags 00;
        input;
        device type PS_32;
}

hidev   1 {
        iopath {cluster 0; eiop 3; channel 032;}
        logical path 0 {flags 00; I_field 00; ULP_id 00;}
        flags 00;
        output;
        device type PS_32;
}

hidev   2 {
        iopath {cluster 0; eiop 3; channel 034;}
        logical path 0 {flags 00; I_field 00; ULP_id 00;}
        flags 00;
        input;
        device type P_8;
}

hidev   3 {
        iopath {cluster 0; eiop 3; channel 036;}
        logical path 0 {flags 00; I_field 00; ULP_id 00;}
        flags 00;
        output;
        device type P_8;
}
```

The following shows the relationship between minor device numbers and `hidev` entries (and therefore physical channel assignments) on Model E systems. This assumes that *himaxpaths* is set to 16.

Path for `hidev`	Minor number range	Dedicated path minor number	Shared paths minor number	`ioctl` only
0	0-15	0	1-14	15
1	16-31	16	17-30	31
2	32-47	32	33-46	47
3	48-63	48	49-62	63

Assume that you want to use channels 034 and 036, cluster 0, and IOP 3 for the TCP/IP interface. If you choose the dedicated path, the minor numbers for the read and the write device are 32 and 48 (0x20 and 0x30). The `hycf` entry is as follows:

```
direct sn2402-hippi     01000006        0020    0030    65448   ;
```

- Model V systems (memory HIPPI systems)

  On Model V systems, the `param` file is not used to configure HIPPI channels. The physical channel number determines the minor device number range associated with a given channel. The lowest numbered channel is the first channel (numbers 0 through 15), with higher number channels being assigned 16 minor device numbers in order. In the following example, assume that two memory HIPPI channel pairs are present:

Channel	Minor number range	Dedicated path minor number	Shared paths minor numbers	Path for `ioctl` only
064	0-15	0	1-14	15
067	16-31	16	17-30	31
0104	32-47	32	33-46	47
0107	48-63	48	49-62	63

On Model V systems, even-numbered channels are always input channels (read devices); odd numbered channels are always output channels (write devices).

A Model V system using the dedicated path for channels 064 and 067 would use minor number 0 for input and 16 (0x10) for output resulting in the following `hycf` entry:

```
direct  sn5194-hippi    01000004        0000    0010    65536   ;
```

2.2.7 Configuring Network Interfaces—GigaRing Based Systems Only

The following sections describe the following steps that you must take to configure each Cray system interface on the network:

1. Define the I/O node configuration
2. Name the Cray interface
3. Choose an Internet address
4. If you are running HIPPI, create an `arp` file, `ghippi#.arp`, where # is the interface number.
5. If you are running Host-to-Host GigaRing, create an `arp`file `gr#.arp`, where # is the interface number.
6. If you are running ATM, create the `atm.pvc` file.

FDDI and Ethernet automatically use the Address Resolution Protocol (ARP) to resolve internet addresses into physical addresses. ATM and HIPPI require you to configure addresses manually.

Note: Before you follow the procedures in this section, make sure that all hardware diagnostic tests have been performed, and that all hardware functions properly. Refer to the appropriate vendor documentation for supporting information and procedures.

2.2.7.1 Defining I/O Node Configuration

Hardware devices that interface with Cray GigaRing based systems are defined in the network portion of the `/opt/CYRIos/snSerialNumber/param` file that resides on the system workstation (SWS). Following is a sample `param` file for a Cray system with GigaRing I/O:

```
network {
     ...
     gether 0 {
          iopath { ring 1; node 4; channel 2; }
          maxinputs 128;
          maxoutputs 128;
     }
     gfddi 0 {
          iopath { ring 1; node 4; channel 0; }
     }
     gatm 0 {
          iopath { ring 1; node 4; channel 3; }
     }
     gr 0 {
          iopath { ring 5; node 7; }
     }
     ghippi 0 {
          iopath { ring 4; node 8; channel 0; }
          maxusers 4;
          maxinputs 80;
          maxoutputs 80;
     }
     ghippi 1 {
          iopath { ring 4; node 8; channel 1; }
          maxusers 3;
          maxinputs 80;
          maxoutputs 80;
     }
          ...
     }
```

The devices are defined according to the following parameters.

- The `iopath` ring number declares the ring the network device resides on.
- The `iopath` node number is the MPN node (for `gether`, `gfddi`, and `gatm`) or SPN node (for `ghippi`), or the local host node for `gr`.
- The `iopath` channel number is the channel on the MPN node, or SPN node.

Note that the channel number is not valid for `gr` interfaces.

An HPN-1 has two HIPPI channel pairs. The first HIPPI channel pair (the top two connectors on the back of the HPN) are channel 0. The second HIPPI channel pair (the bottom two connectors on the back of the HPN) are channel 1.

An HPN-2 has either a single 100 Mb of 200 Mb HIPPI channel pair. Both are designated as channel 0.

2.2.7.2 Naming the Cray GigaRing Interface

The interface name, which TCP/IP uses to access a given hardware device, consists of two parts. The first part, which is the interface name prefix, indicates the type of hardware. The second part, which is the number of the interface name (also known as the interface number), identifies the physical device.

2.2.7.3 Interface Name Prefix

The interface name prefix is derived form the UNICOS driver that controls the hardware device. This prefix indicates the major number that TCP/IP uses when it defines the character special device for the given hardware. Following is a list of the prefixes and a description of the associated interfaces:

Prefix	Type of interface
`gatm`	Asynchronous Transfer Mode (ATM)
`gether`	Ethernet
`gfddi`	FDDI
`ghippi`	HIPPI
`gr`	Host-to-Host GigaRing

2.2.7.4 Identifying Character Special Devices

You do not need to define character special device nodes to use TCP/IP with Ethernet, ATM, HIPPI, or FIDDI channels; the host kernel creates the I-node(s) it needs. TCP/IP is hard coded to use user number zero. However, if you are using the HIPPI channel in “raw” mode, you must define character special device nodes.

On non-GigaRing I/O Cray HIPPI implementations, there are separate character special device files for input and output. The GigaRing I/O HIPPI “raw” character special devices are full-duplex: each device logically represents a HIPPI input and output channel. For example:

```
fd = open("/dev/ghippio/u0", O_RDWR);
/*  reades and writes use the same file descriptor (fd)  */
```

For GigaRing networking I/O, there is a single host driver for all networking interfaces; `gether`, `gfddi`, `gatm`, `gr`, and `ghippi`. The *major* device number is always 25. The *minor* device number for HIPPI character special devices for a specific interface depends on the order in which all the network devices (not just the HIPPI interfaces) are defined in the host’s configuration file.

The number of character special device files for a HIPPI interface should at least match the number of users defined by maxusers in the host's configuration file. Each device file corresponds to a specific user of the interface. Multiple users can share the same HIPPI interface at the same time. The HIPPI channel can be used in "raw" mode and by TCP/IP at the same time. The maximum number of users for any networking interface is 256 (user number 0 to 255).

GigaRing character special devices have the following format:

`/dev/`*gdevicenameX*`/`*uY*

Where *X* is the HIPPI device number and ranges from 0 to # of interfaces less 1 (that is, *N*`-1`, where *N* = *number of interfaces*), *devicename* is the channel type, an *Y* is the user number (or logical path) ranging from 0 to maxusers –1

For HIPPI character special device usage, there are restrictions or capabilities associated with a particular user path, i.e., a specific path is not the dedicated path or has a ULPid associated with it. If TCP/IP is being used on the HIPPI channel, it will always use user path 0 (`/dev/ghippix/u0`).

2.2.7.5 Creating HIPPI Character Special Device Nodes

You can determine the order that the networking interfaces are defined by either perusing the host's configuration file or looking at the output from `netstat -i`.

For interfaces defined in the following order, the character special device nodes for the two HIPPI interfaces are as follows:

```
Interface   maxusers   Starting minor#
---------   --------   ---------------
gether0     n/a             0
gfddi0      n/a           256
gatm0       n/a           512
gr0         n/a           768
ghippi0      4            1024
ghippi1      3            1280


mkdir /dev/ghippi0
chmod 755 /dev/ghippi0
cd /dev/ghippi0
/etc/mknod u0 c 25 1024
/etc/mknod u1 c 25 1025
/etc/mknod u2 c 25 1026
```

```
/etc/mknod u3 c 25 1027
chmod 666 u*

mkdir /dev/ghippi1
chmod 755 /dev/ghippi1
cd /dev/ghippi1
/etc/mknod u0 c 25 1280
/etc/mknod u1 c 25 1281
/etc/mknod u2 c 25 1282
chmod 666 u*
```

Warning: If, at a future time, additional networking interfaces are added to the host's configuration file, they must come after the HIPPI interface definitions. If they are placed in front of the HIPPI interface definitions, the HIPPI interface minor device numbers increase by 256 and become invalid.

2.2.7.6 Choosing an Internet Address

Each interface on your system must have an Internet address. The network and subnet parts of the Internet address of the interface must correspond to the Internet address of the network to which the interface is attached. The host portion of the address can be selected by using the preferred administrative practices for your site. (Some sites have a central network administrator who determines the new Internet addresses for new systems and interfaces; other sites let the individual system administrator decide.)

When you are choosing an Internet address for an interface that is attached to a new TCP/IP network, see Section 2.2.4, page 30, for guidelines.

2.2.7.7 Creating the `ghippi#.arp` File

Note: For software loopback interfaces, direct FDDI interfaces, Ethernet interfaces, and ATM interfaces, it is not necessary to create an `arp` file.

The information you put in this file is used as input to the `arp`(8) command, which initializes the interface with hardware address information. This configuration file, sometimes known as the `arp` file, varies according to the type of network hardware used. Each ghippi connection requires a separate `arp` file.

Use the following format when you create the `ghippi#.arp` file for HIPPI connections:

```
hostname 00:00:AA:BB:CC:DD
```

hostname	Host name that appears in the `/etc/hosts` file, or Internet address expressed in decimal notation.
ifield (AA:BB:CC:DD)	HIPPI I-field value for connection to this host (hexadecimal notation) where AA:BB:CC:DD corresponds to the I-field 0xAABBCCDD. The I-field is typically used to make a HIPPI connection through one or more HIPPI crossbar switches. The I-field is used by the switch(es) to determine the path that is required to complete the connection. In the most simple case, the I-field contains the camp-on bit plus the port number to which the host is connected. The camp-on bit is bit 2**24 (0x01000000). This bit directs the switch to keep trying to make the connection until the connection is completed or the source abandons the connection attempt.

Some switches support additional routing implementations in which the destination address is not a simple port number but is instead a value that the switch converts into one or more port numbers. Consult the switch manufacturer's documentation for details on constructing an appropriate I-field.

If a double-wide (64-bit) HIPPI connection can be made between two systems, the I-field must include bit 2**28 (0x10000000) for the driver to make use of this capability. If this bit is not set in the I-field, data is sent in 32-bit mode even though the 64-bit capability exists. Only HPN-2 can be configured to support the 64-bit HIPPI interface. Some switches and HIPPI fiber optic channel extenders offer 64-bit capability as well. If a switch offers both 64-bit and 32-bit capabilities, the `ghippi#.arp` file should use the double-wide bit for hosts to which a 64-bit connection can be made. If a host has double-wide capability, but another host is only 32-bit capable, the entry for this host should not have the double-wide bit set. The driver ensures that the connection is made

	in the appropriate mode on a per-packet basis. The double-wide bit is also significant in the case in which no switch is present.

The following is an example of a `ghippi#.arp` file for a HIPPI connection:

```
    sn1001-hippi  00:00:03:00:00:00
    sn1701-hippi  00:00:03:00:00:01
    sn1601-hippi  00:00:03:00:00:02
    uss-hippi     00:00:03:00:00:03
    sn1061-hippi  00:00:03:00:00:04
```

2.2.7.8 Creating the `atm.pvc` File

The `atm.pvc` file is used for configuring the Permanent Virtual Circuit (PVC) identifiers for all GigaRing ATM interfaces.

The `spansd` implements the FORE Systems proprietary ATM signalling protocol called SPANS. The `q2931d` implements the standard UNI ATM signalling protocol. Both daemons should be in the `SYS2` group, and are `/etc/spansd` and `/etc/q2931d`, respectively.

Use the following format to create the `atm.pvc` file for ATM connections:

```
hostname ifc AAL VPI VCI QOS
```

The following is an example `atm.pvc` file for an ATM connection:

```
# hostname   ifc    AAL   VPI   VCI   QOS
# --------   ---    ---   ---   ---   ---

atmhost1     gatm   5     0     32    5000
atmhost2     gatm   5     0     32    0
```

Where:

hostname	Name of the remote host that this system communicates with over the ATM network. This name must appear in the host database (i.e., `/etc/hosts` file).
ifc	Network interface. Must be `gatm` for a GigaRing based system.
AAL	ATM Adaption Layer Type. Defines the layer type to use when communicating with the specified

	host. Only AAL 5 is currently supported. Specify this number in decimal.
VPI	Virtual Path Identifier. Identifies the path to use when communicating with the specified host. The VPI is placed into each ATM cell header so that the cell can be routed through the ATM network. Currently, GigaRing ATM interfaces support only VPI 0. Specify this number in decimal.
VCI	Virtual Circuit Identifier. Identifies the circuit to use when communicating with the specified host. The VIC is placed into each ATM cell header so that the cell can be routed through the ATM network. This number should be between 32 and 1023. Consult your local network administrator to determine the VCI. Specify this number in decimal.
QOS	Quality of Service in Kbps. Determines the peak data rate (expressed in kilobits per second) that this host will deliver ATM calls to the remote host via the GigaRing ATM interface. Placing a zero in this field disables the peak rate control feature when sending to the specified host, allowing unlimited bandwidth.

2.2.7.8.1 Creating the `gr#.arp` File

The information you put in this file is used as input to the `arp` command, which initializes the interface with hardware address information. This configuration file, sometimes known as the `arp` file, varies according to the type of network hardware used. Each `gr` interface requires a separate `arp` file. There needs to be an entry in the `arp` file for each host you wish to access via the `gr` interface.

Use the following format when you create the `gr#.arp` file for Host-to-Host GigaRing interfaces:

```
hostname 00:00:00:00:AA:BB
```

hostname	Host name that appears in the `/etc/hosts` file, or Internet address expressed in decimal notation.
ifield (00:00:00:00:AA:BB)	Where *AA* is the ring number for this host (hexadecimal notation). This ring number will

be the same value as that specified in the param file for this `gr` interface. (All entries in a specific `gr#.arp` file will have the same ring number.)

Where *BB* is the node number for this host (hexadecimal notation).

The following is an example of a `gr#.arp` file:

```
sn9132-gr      00:00:00:00:05:01
sn7025-gr      00:00:00:00:05:04
```

2.2.8 Configuring Daemons

The following daemons can be configured:

Daemon	Configuration
`gated`(8)	Performs dynamic routing
`lpd`(8)	Spools print files to remote printers
`named`(8)	Provides domain name service
`sendmail`(8)	Performs Simple Mail Transfer Protocol (SMTP) operations
`snmpd`(8)	Performs Simple Network Management Protocol (SNMP) operations
`ntpd`(8)	Provides mechanisms to synchronize time and coordinate time distribution
`inetd`(8)	Provides a sublist of daemons that can be started

Note: The `gated`(8), `named`(8), `snmpd`(8), and `ntpd`(8) daemons must not be configured when you are running the Cray ML-Safe configuration of the UNICOS system.

These daemons are described in the following sections.

2.2.8.1 The `gated` Daemon

As an alternative to the static routing that is performed by the `route`(8) command, the `gated`(8) daemon can oversee dynamic management of the routing table on the local Cray host. The `gated` daemon communicates with remote hosts by using one or more routing protocols. Using these protocols, `gated` collects information that enables it to determine and install the correct routes that it uses to achieve optimal routing for packets that originate from the local Cray host. The `gated` daemon has the simple ability to detect the failure of directly attached networks or gateways and to manipulate the routing table on the local Cray host to reestablish IP service to remote hosts and networks whose path to the local Cray host is affected by an unavailable network or gateway.

The `gated` daemon supports RFCs 1388, 1583, 1058, and parts of 1009. The specific routing protocols supported by `gated` are as follows:

Protocol	Description
RIP and OSPF	Two interior protocols that are used to exchange routing information within commonly administered local networks

You can configure the `gated` daemon independently in each of the supported protocols to supply routing information, to listen for routing information, or both. For example, the `gated` daemon can be configured to simultaneously listen for and supply OSPF routing information, and to only listen for RIP routing information.

2.2.8.1.1 Configuration Guidelines for `gated`

The actual routing protocols that you configure for the `gated` daemon to use on your Cray system depend on the routing protocols that the remote hosts and gateways use on the networks to which the system is attached. For example, if the other hosts on an attached network are using the RIP protocol to exchange routing information, the `gated` daemon on the local Cray system should also be configured to use the RIP protocol to exchange routing information. However, the following general guidelines for configuring `gated` apply to Cray systems:

- A Cray system can have more than one directly attached network, and it can be considered a gateway system between its directly attached networks, but it is uncommon for a Cray system to function intentionally as a generic gateway. A Cray system usually serves as an endpoint for connections to and from the system, which allows the resources of the system to be spent on user tasks rather than forwarding packets that are intended for other systems. Consequently, it is more functional to configure `gated` to provide dynamic rerouting when it encounters network or gateway failure than it is to configure `gated` to use the Cray system as a true gateway.

  Note: The Cray system must not perform forwarding (that is, act as a gateway) when it is running the Cray ML-Safe configuration of the UNICOS system.

- Cray systems typically do not serve as exterior gateways for their local networks; therefore, they do not support Exterior Gateway Protocol (EGP) or Border Gateway Protocol (BGP) protocol information.
- The gateways on networks that are directly attached to the Cray system must be configured not only to send updates to the Cray system, but to work with the configuration on the Cray system, because proper dynamic routing requires cooperation among hosts to exchange routing updates. Configuring `gated` on the Cray system in isolation from the configuration of the gateway system is generally not sufficient to achieve optimal routing.
- Cray systems supports directly attached network media with no broadcast capability. Some routing protocol implementations rely on a broadcast capability for the local network to supply routing updates to other hosts (such as the Berkeley `routed` program, which implements RIP). Therefore, any gateway that wants to supply routing updates over a nonbroadcast media to a Cray host must be capable of being configured to supply routing updates over nonbroadcast network media. Usually, this means that the gateway has implemented the `gated` daemon as a routing protocol, but other routing protocol implementations can exist on a given gateway. Consult the documentation supplied by the vendor of your gateway system to plan your dynamic routing capabilities.

2.2.8.1.2 The `/etc/gated.conf` File

The complete format of the `/etc/gated.conf` file is described in the `gated-config`(5) man page. If you are using the UNICOS ICMS, configure this file by using the `Configure System -> Network configuration -> TCP/IP configuration -> Routing` menu. Some general guidelines for configuration of this file are as follows:

- You can specify hosts and networks in the `gated.conf` file by using either the name or the Internet address. However, if you are using the `named` daemon or the `resolv.conf` resolver library, you should use only Internet addresses in the `gated.conf` file.
- When you do not want to supply routing information to other gateways, but want `gated` only to listen for RIP updates, you must specify `nobroadcast` on the appropriate `rip` statement.
- Any gateways residing on nonbroadcast networks to which you want to supply RIP updates must be explicitly specified by the `sourcegateways` directives of the appropriate `rip` statement because of the lack of a broadcast capability on the directly attached networks.

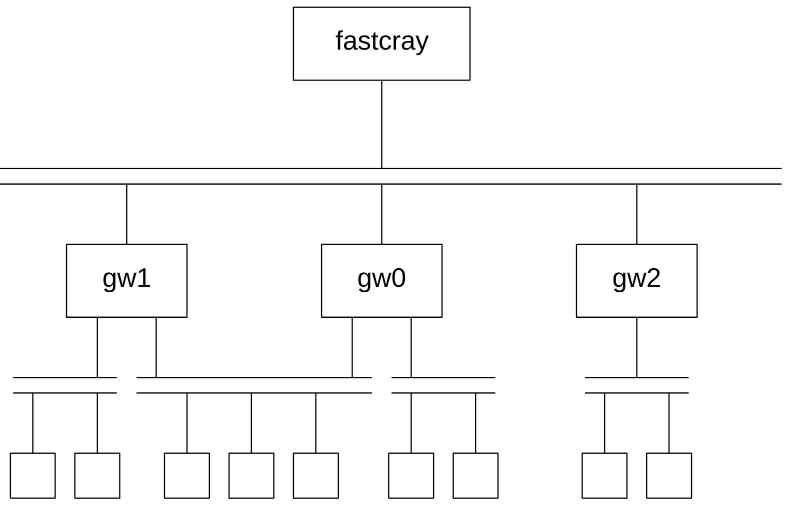
- You can exclude the interfaces to network media that function as virtual point-to-point links, such as HIPPI or FEI-3 interfaces by using the `interface` *interface_list* `noripout` directive of the appropriate `rip` statement.
- To configure static routes, which are never removed in response to routing protocol updates, use the `static` statement.

  Note: extensions to the static route options in the `gated` configuration file provide access to all of the proprietary routing extensions, such as per-group restricted routing, and type of service routing; see `gated-config`(5) for details.
- When a gateway on a network that is directly attached to the Cray system can redirect traffic by sending `ICMP REDIRECT` messages, you can specify that gateway as the gateway for the `default` route in the `static` statement.
- You can direct `gated` to listen for a specific set of hosts or networks through the use of `import` statements. Information received in routing updates about other hosts or networks is ignored. Alternatively, use of `import` statement `restrict` directives excludes a specific set of hosts or networks from consideration.
- You can limit the information that `gated` supplies to other gateways to a specific set of hosts or networks through `export` statements. Alternatively, use of `export` statement `restrict` directives causes `gated` to supply information to other gateways about all hosts or networks that are not included in the specified set.

2.2.8.1.3 `gated` Configuration Examples

This section contains two examples of `gated` configuration.

Consider the following sample network configuration:



a10258

The gateway `gw0` of this network configuration serves as the default gateway for connections to the Cray system `fastcray`. Instead of sending all network traffic through this one default gateway, however, it is recommended that traffic for the networks that are attached to the other two gateways `gw1` and `gw2` use those gateways. (For a more complete discussion of the effect of network routing on overall network performance, see Section 2.3.3, page 148.) When `gw0` can send `ICMP REDIRECT` messages, the following sample `/etc/gated.conf` file is sufficient to ensure proper maintenance of the routing tables on `fastcray`:

```
rip no ;

redirect yes ;

static {
default gateway gw0 ;
} ;
```

The `redirect yes` statement in this sample `gated.conf` file indicates that `fastcray` listens to `ICMP REDIRECT` messages from the gateways on its attached network and installs them in the kernel routing tables. (Conceptually, this occurs in response to an `ICMP REDIRECT` message; however, in reality, the UNICOS kernel installs the route directly, and the `redirect yes` statement prevents `gated` from removing those installed routes.)

For this configuration, when sending traffic to a host that is accessible through the gw2 gateway, fastcray sends the initial packet to its default gateway gw0. Like any gateway, gw0 consults its routing table and forwards the packet to gateway gw2, and the packet eventually reaches its destination host on one of the networks that is attached to gateway gw2. After forwarding the packet for delivery, gateway gw0 determines that because fastcray and gateway gw2 are both on the attached network from which gateway gw0 received the packet, it is more efficient for fastcray to forward packets intended for the destination host directly to gw2 (where gateway gw0 would send the packets anyway). Then it sends an ICMP REDIRECT message to fastcray, informing it that packets intended for that destination host should be redirected to gateway gw2.

One problem with this simple configuration is that some gateway systems send a separate ICMP REDIRECT message for each appropriate destination host instead of sending one message for the entire network. This can possibly lead to inefficient use of system resources on the Cray system, because each ICMP REDIRECT message generates a new host-level entry in its routing table. Also, the procedure of sending an initial packet to gw0 for redirection and processing the resultant ICMP REDIRECT message adds a small amount of overhead to establishing initial connections to each destination system. You can eliminate this problem by using static network routes on the Cray system. This can be accomplished by changing the static statement in the /etc/gated.conf file to the following:

```
static {
        default gateway gw0 ;
        othernet1 gateway gw1 ;
        othernet2 gateway gw2 ;
} ;
```

By adding static routes for othernet1 and othernet2, fastcray sends packets for those networks directly to gw1 and gw2, and it eliminates the need for gw0 to send ICMP REDIRECT messages for the traffic to those networks.

The following sample network is a more complex example than the previous one:

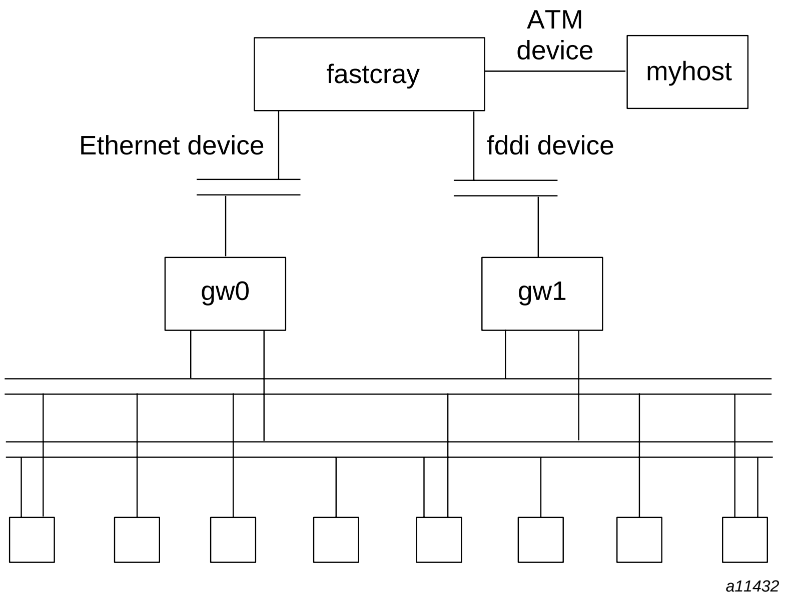


Figure 7. Sample network configuration

This network configuration might be appropriate in a situation in which a network administrator wants to provide two (or more) gateways to the Cray system to ensure access to the system when one gateway fails. This technique requires proper configuration of not only `gated` on the Cray system but also the equivalent dynamic routing programs on the directly attached gateways.

The `gated` daemon detects failure of a gateway by tracking the elapsed time since individual routes to destination hosts or networks were received in RIP updates. If 180 seconds elapse without mention of a route in a routing update, the route is deleted and considered to be in a *hold down* state. If the route is being exported, for the next 120 seconds the route is announced with a metric of infinity. This causes all listening routers to stop using this host for the route. This means that the `gated` daemon's detection of a gateway failure is implicit; when no updates are received from the failed gateway, the routes through that gateway are held down and deleted. This allows other routes to the destinations to be installed.

This method of failure detection means that unless otherwise configured, `gated` assumes that a gateway that does not send routing updates is down when in fact it might be up but not configured to send updates.

The following example is of a `/etc/gated.conf` file. This example provides a basic back-up mechanism for this network configuration when a failure of either gateway occurs.

For Model E based systems, where `np0` is an Ethernet device, `np1` is a FDDI device, and `np2` is an ATM device:

```
interfaces   {
   interface np2 passive ;

};
   rip yes {
         nobroadcast ;
         sourcegateways gw0 gw1 ;
   } ;

   export proto rip gateway gw0 {
          proto direct interface np1 np2 ;
   } ;

   export proto rip gateway gw1 {
        proto direct interface np0 np2 ;
   } ;
```

For GigaRing based systems, where `gether0` is an Ethernet device, `gfddi0` is a FDDI device, and `gatm0` is an ATM device:

```
interfaces   {
   interface gatm0 passive ;

};
   rip yes {
         nobroadcast ;
         sourcegateways gw0 gw1 ;
   } ;

   export proto rip gateway gw0 {
          proto direct interface gfddi0 gatm0 ;
   } ;

   export proto rip gateway gw1 {
        proto direct interface gether0 gatm0 ;
   } ;
```

There are several items in this example to note:

- The `interface np2/gatm0 passive` statement prevents `gated` from deleting routes to the OWS/SWS, even when it received no routing updates from the OWS/SWS.

- The two `export` statements inform each gateway of not only the interface through which `fastcray` communicates with the OWS/SWS (`np2/gatm0`), but also the interface through which the Cray system communicates with the other gateway. This means that each gateway recognizes the address of the other interface on the Cray system and forwards packets for that address directly to the Cray system, instead of routing them to the gateway.

- The `export` statements instruct `gated` to advertise only the other directly attached interfaces to `fastcray`. Implicitly, this means that in the routing updates it distributes, `gated` does not advertise any routes that it learned through an update from either gateway. This helps prevent `fastcray` from functioning as a generic gateway.

  For example, the following `export` statement that contains an added `proto rip` directive makes the Cray system function as a generic gateway by instructing `gated` to advertise routes it learned through RIP updates from either gateway and also to explicitly advertise the networks for the `np0/gether0` and `np2/gatm0` interfaces:

  For IOS–E based systems, where `np0` is an Ethernet device and `np2` is an ATM device:

  ```
  export proto rip gateway gw1 {
          proto rip ;
          proto direct interface np0 np2 ;
  } ;
  ```

  For GigaRing based systems, where `gether0` is an Ethernet device and `gfddi0` is an ATM device:

  ```
  export proto rip gateway gw1 {
          proto rip ;
          proto direct interface gether0 gfddi0 ;
  } ;
  ```

- Although this configuration provides a general backup mechanism if a gateway fails, note that this backup does not preserve connections that exist at the time of the failure. When a gateway fails, any connections that are using that gateway are severed. However, the dynamic routing capabilities of `gated` switch the routing to the alternate gateway, which allow users to reestablish connections to `fastcray` and to continue use while the cause of the gateway failure is investigated and corrected.

2.2.8.2 The `lpd` Daemon

When the system is initialized, the line printer daemon passes through the `/etc/printcap` file to obtain information about the existing printers; it prints any files that are queued. In subsequent operations, this daemon listens for and processes line printer requests that come in on port `515` (see Section 2.2.5.1, page 34).

The `/etc/printcap` file configures all printers that are available to users executing on the UNICOS system. The following commands provide users the indicated access to those printers defined in `/etc/printcap`:

Command	Description
`lpq`(1B)	Displays the files queued for printing.
`lpr`(1B)	Spools a file for printing.
`lprm`(1B)	Removes a file from the queue. If printing has already started on the file, it is aborted.

The following commands provide control of the state of each printer:

Command	Description
`lpc`(8)	Allows the super user to change the status of and control the queues for each printer.
`lpd`(8)	Controls the printing of all files that were spooled for printing by the `lpr` command.

To configure and make printers available to users running on the UNICOS system, the following steps must be performed:

1. Ensure that `lpd` is started during system initialization.
2. Create the `/etc/printcap` file, using the rules listed in the following section, " `printcap` file creation rules."
3. Configure the printer spool directories, as listed in the `/etc/printcap` file.

2.2.8.2.1 `printcap` File Creation Rules

This section lists the rules for creating the `/etc/printcap` file. A sample `printcap` file is shown on Section 2.2.8.2.4, page 76.

- Each line in the file defines one printer.

- All fields contained on a line define specific characteristics that are associated with the printer. Each field is delimited by a colon (`:`).
- Spaces are significant and taken literally, rather than ignored.
- Any data following a # and continuing to the end of the line is considered a comment.
- The first field contains a list of names by which a given printer is known to users who are executing on UNICOS. Each name in the list must be delimited by a vertical bar (`|`). All printer commands (`lpq`, `lpr`, `lprm`) accept the `-P` option, or use the `PRINTER` environment variable, to indicate a name in this list.
- The remaining fields are identified by keywords and can be arranged in any order except where otherwise indicated. If a keyword is not specified, or is followed by a `@`, the default value is used. The following section, "`printcap` file keywords," lists the available keywords.

2.2.8.2.2 `printcap` File Keywords

The `/usr/src/net/tcp/usr/ucb/lpr/lp.local.h` file contains the default value for each `printcap` file keyword. Only the first occurrence of a keyword appearing on any printer definition line is used. The following example `printcap` file is built with the `lp` command and defines default values.

```
/*
 * Defaults for line printer capabilities data base
 */
#define DEFLP           "lp"
#define DEFLOCK         "lock"
#define DEFSTAT         "status"
#define DEFSPOOL        "/usr/spool/lpd"
#define DEFDAEMON       "/usr/lib/lpd"
#define DEFLOGF         "/dev/console"
#define DEFDEVLP        "/dev/lp"
#define DEFRLPR         "/usr/lib/rlpr"
#define DEFBINDIR       "/usr/ucb"
#define DEFMX           1000
#define DEFMAXCOPIES    0
#define DEFFF           "\f"
#define DEFWIDTH        132
#define DEFLENGTH       66
#define DEFUID          1
```

The following keywords have the indicated meaning and default:

`ff=`	Indicates the character string that represents a form feed to the indicated printer. This keyword is used only when the printer is defined as being locally attached. The `DEFFF` constant defines the form feed default value, which is set to `"\f"`.
`if=`	Specifies the name of an input filter program that is executed for each file that is printed. Standard input to this program is set up to receive the control file, and then the actual file to be printed. Standard output is set up to go to the actual printer device entry (defined by `lp=`). This keyword is used only when the printer is defined as being locally attached. There is no default for this keyword; if it is not specified, no input filter is started.
`lf=`	Specifies the log file that the `lpd` daemon creates to be used for error messages that are encountered while printing a file on the given printer or sending a file to the remote system for printing. The `DEFLOGF` constant defines the log file default value, which is set to `/dev/console`.
`lo=`	Indicates the default lock file name. This file exists in the spool directory for the given printer. Its contents indicate the process ID of the daemon started up to control printing. This file also contains the control file name of the current file being printed, if one is currently active. The file mode of the default lock file indicates whether printing is enabled (that is, if the `owner execute` bit is off), whether queuing is enabled (that is, if the `group execute` bit is off), and whether `lpd` should reorder its queue after it finishes printing the current file (that is, the `world execute` bit is on).

The `DEFLOCK` constant defines the lock file default name, which is set to `lock`. The value of the `sd=` parameter controls the path name to this file.

`lp=`	Indicates the printer device to be opened when a file is requested to be printed on the indicated printer. If specified as `lp=:`, the printer is assumed to be located on a remote host. The `DEFDEVLP` constant defines the printer device default value, which is set to `/dev/lp`.
`ma`	Specifies the maximum security label allowed on this printer. This keyword is meaningful only with UNICOS security.
`mi`	Specifies minimum security label allowed on this printer. This keyword is meaningful only on with UNICOS security. **Note:** With UNICOS security, the `ma` and `mi` keywords define the maximum and minimum labels supported by the printer. If the printer is remote, it must be trustworthy to print between those labels, and must supply banner page and per-page human-readable labels on the printed output.
`of=`	Specifies the name of an output filter program that is executed each time the `lpd` daemon starts up. If another filter program is being used with a given, locally attached printer (for example, an input filter), this output filter performs whatever processing is required by the printer between print files. If no other filter program is being used with a given, locally attached printer, this output filter also acts as an input filter (as described under the `if=` keyword). For remote printers, standard input to this program is set up to receive control information, the print file's control file, and the actual file to be printed. Standard output is set up to go to the remote system. There is no default for this keyword; if it is not specified, no output filter is started.

The control information passed to the output filter, when used for remote printers, is as follows:

Information	Description
`3` *number filename*	Indicates that the information following this line is the print file's control file. *number* is the size (in bytes) of the control file being sent. *filename* is the name of the control file being sent.
`2` *number filename*	Indicates that the information following this line is the actual file to be printed. *number* is the size (in bytes) of the print file being sent. *filename* is the name of the print file being sent.
`1`	Indicates that an error occurred while the current file was being read; the file is not printed. This also signifies the end of the current file.
`0`	Indicates the end of the file currently being sent to the remote system.

`rm=` Indicates the remote host name (or valid alias) of the host to which the printer is attached. The value specified for this keyword takes precedence over what is specified for the `lp=` keyword. Therefore, when this name is different from the official host name (or valid alias) of the local host, the `lp=` keyword is ignored, and the file is sent to the remote host.

There is no default for this keyword; if it is not specified, the printer is assumed to be locally attached.

`rp=`
: Indicates the remote printer's name. This is the name by which the printer is known on the remote host (that is, one of the names in the first field of the remote host's `printcap` file).

  The `DEFLP` constant defines the remote printer name default value, which is set to `lp`.

`sd=`
: Specifies the full path name of the directory to be used for spooling files for printing. This directory also contains all of the control files (that is, the `lock` and `status` files) for the given printer.

  Note: With UNICOS security, if the printer supports a range of security labels, its spool directory must be created as a multilevel directory.

  The `DEFSPOOL` constant defines the path of the spool directory default value, which is set to `/usr/spool/lpd`.

`st=`
: Indicates the default status file name. This file exists in the spool file directory. Its content describes the current status of the line printer daemon (`lpd`) while it is running.

  The `DEFSTAT` constant defines the default status file's default name, which is set to `status`.

`tc=`
: Specifies another printer's name and is used to indicate that this printer definition also contains the list of keywords identified on the line defining the other printer. If specified, this must be the last keyword in the printer's definition; otherwise, it is ignored.

  There is no default for this keyword; if it is not specified, the printer's characteristics are assumed to be totally defined by the given line.

`tr=`
: Indicates the character string to be printed after all queued files are printed. This keyword is

used only when the printer is defined as being locally attached.

There is no default for this keyword; if it is not specified, nothing is sent to the printer when all queued files are printed.

2.2.8.2.3 Remote Printers and the UNICOS System

Remote printers that will receive output from a UNICOS system are expected to operate within the following restrictions:

- If the remote printer will print output at a single label, it is acceptable to print this labeled output on prelabeled paper.
- If the remote printer is expected to print multiple labels of output, the printer must be attached to a system that supports labeled printing.
- Any remote multilabel printer server must interpret the label of a connection on the `lpd` port as the label to be printed on the output.
- Labels printed on output must be human-readable.

As described in the previous section, the `mi` and `ma` keywords in the `/etc/printcap` entry for a particular printer define the range of labels allowed on output transmitted to that remote printer.

2.2.8.2.4 Sample `printcap` File

The following is a sample `printcap` file:

```
# Example file:  /etc/printcap
#
lp0|myprinter:rm=remote1:sd=/usr/spool/printers/lp0:
lp1|devprinter:rm=remote2:rp=ps1:sd=/usr/spool/printers/lp1:
lp2:lp=:rm=remote3:rp=hisprinter:sd=/usr/spool/printers/lp2:
lp3:lp=/dev/null:if=/etc/myfilter:sd=/usr/spool/printers/lp3:
lp4:sd=/usr/spool/printers/lp4:tc=lp3:
```

In the preceding example, each printer has the following characteristics:

- `lp0`
  - Known also as `myprinter`.
  - Attached to the remote system `remote1`.

  - Known as `lp` to `remote1` because no `lp=` keyword was specified.
  - Queues all files to be printed in the `/usr/spool/printers/lp0` directory before being sent to the remote system.

- `lp1`
  - Known also as `devprinter`.
  - Attached to the remote system `remote2`.
  - Known as `ps1` to `remote2`.
  - Queues all files to be printed in the `/usr/spool/printers/lp1` directory before being sent to the remote system.
- `lp2`
  - Attached to the remote system `remote3`.
  - Known as `hisprinter` to `remote3`.
  - Queues all files to be printed in the `/usr/spool/printers/lp2` directory before being sent to the remote system.
- `lp3`
  - Set up to execute the filter program `/etc/filter`. All files are discarded after the filter program performs its processing on them (`lp=/dev/null`).
  - Queues all files to be printed in the `/usr/spool/printers/lp3` directory before being discarded.
- `lp4`
  - Queues all files to be printed in the `/usr/spool/printers/lp4` directory before being discarded.
  - Contains all other printer characteristics that are identical to those of `lp3`.

2.2.8.3 The named Daemon

Note: The use of the `named` daemon is not allowed with the Cray ML-Safe configuration of the UNICOS system.

As an alternative to using the `/etc/hosts` file, you can configure As an alternative to using the `/etc/hosts` file, your UNICOS system to use the local domain name server, `named`(), or the associated domain name resolver routines in the system library, to provide information (for example, host names and Internet numbers) about remote hosts that are accessible to the local Cray host.

Because `named` is not available on systems running UNICOS security during startup, every name that appears in the `/etc/config/spnet.conf` file must exist in the `/etc/hosts` or `/etc/networks` files. This is because `spnet`(8) must be executed to create the network access list (NAL) before applications can begin creating socket connections.

2.2.8.3.1 Setting up the Cray System As a Name Server

To use the `named` server, first create an empty `/etc/hosts.usenamed` file on your Cray system. This file does not contain any text because the presence of the file indicates that the domain name service is to be used. Consequently, all queries for information concerning hosts on the network will be resolved by the domain name service, rather than by looking in the `/etc/hosts` file.

However, you still need the `/etc/hosts` file to resolve names encountered during startup (for example, names in the `/etc/config/interfaces` file).

The `/etc/named.boot` file contains start-up information for the `named` server. When this file includes a line beginning with the keyword `directory`, `named` uses the specified directory as its current directory. Files that `named` must consult when starting up or while running are then read from or placed in the designated directory.

The `named` server always runs as a caching server. In addition, the `named` server can be run as a slave, caching-only, or master server. Each of these processes is described in the following sections.

The `named.boot` file must always specify, on a line beginning with the keyword `primary`, primary server authority for the reverse-address mapping domain (`in-addr.arpa`) that corresponds to the local loop-back interface (see the following section for more information on primary servers).

During operating system boot and before the network interfaces are configured, `named` cannot access other DNS servers. Therefore, DNS must be temporarily disabled; for more information about how to do this, see the `named`(8) man page.

See the *Name Server Operations Guide for BIND*, written by Kevin J. Dunlap and Michael J. Karels, for more information on name service.

2.2.8.3.2 named As a Slave Server

Internet addresses of one or more forwarding servers. Finally, it must If you plan to run named as a slave server, in addition to the directory line, the /etc/named.boot file must also specify, on a line beginning with the keyword forwarders, the contain a line consisting of the keyword slave. The following is a sample /etc/named.boot file for a slave server:

```
# /etc/named.boot file for slave server

directory    /usr/named
forwarders   123.45.67.89   234.56.78.90
primary      0.0.127.in-addr.arpa localhost.rev
slave
```

When the /etc/named.boot file is configured for a slave server, and it lists one or more forwarding servers to resolve recursive queries, all queries that cannot be answered from the local server's cache of responses are forwarded to the forwarding servers until one of the forwarding servers resolves the query. Responses are cached by the local server, resulting in faster responses to queries and less network traffic than the time and traffic associated with the remote server method (see Section 2.2.8.3.5, page 81).

With this method, direct access to root domain name servers on the Internet is not required. However, this method requires a remote server that can resolve recursive queries.

2.2.8.3.3 named As a Caching-only Server

If you plan to run named as a caching-only server, in addition to the directory line, the /etc/named.boot file must designate, on a line beginning with the keyword cache, a zone file that contains information about the root domain name servers on the Internet. A *zone* is a delegated subset of the domain name service tree. It contains the tree that is specified by the domain name minus subtrees delegated to other zones. It is the portion of the domain name service tree under a single administrative control. A *zone file* is a text file that contains information about a given zone.

The following is a sample /etc/named.boot file for a caching-only server:

```
# /etc/named.boot file for caching-only server

directory    /usr/named
cache        .                      root.cache
primary      0.0.127.in-addr.arpa   localhost.rev
```

In the preceding example, the third field on the lines that begin with the keywords cache and primary designate the files root.cache and localhost.rev (in the /usr/named directory) as zone files; root.cache contains the Internet addresses of and information about the root domain name servers on the Internet, and localhost.rev contains the information that refers the special Internet address 127.0.0.1 to the local Cray host.

The caching-only method does not require (but can use) a remote forwarding server that can resolve recursive queries. It does require direct access to the root domain name servers on the Internet.

2.2.8.3.4 named As a Master Server

If you plan to run named as a master server, you will need all of the information specified in the /etc/named.boot file for a caching-only server, and also additional lines beginning with the keywords primary or secondary for any zones over which the local named server has primary or secondary authority.

A *primary master server* for a zone loads the data for that zone from a zone file on disk (specified in the /etc/named.boot file). The data in the zone file consists of the authoritative host names, Internet addresses, and other information for the zone.

A *secondary master server* for a zone is delegated authority for the zone by the zone's primary master server, and periodically updates the data for the zone as needed. This data can also be stored in a back-up copy on disk to provide uninterrupted service if the primary master server for the zone is unavailable and the local server has initialized its data.

The following is a sample /etc/named.boot file for a master server:

```
# /etc/named.boot file for master server

directory    /usr/named
cache        .                            root.cache
primary      0.0.127.in-addr.arpa         local
primary      ourdomain.com                ours
primary      12.in-addr.arpa              ours.rev
secondary    theirdomain.edu              theirs.bak
secondary    78.56.234.in-addr.arpa       theirs.rev.bak
```

In the preceding example, the last four lines designate the master files (ours, ours.rev, theirs.bak, and theirs.rev.bak), which contain information about the associated zones (ourdomain.com, 12.in-addr.arpa, theirdomain.edu, and 78.56.234.in-addr.arpa, respectively). The master files for the zones over which this server has primary authority (`ours` and `ours.rev`) must contain the actual information about the hosts and Internet addresses in the network; that is, these are the files that must be updated if a host is added to or deleted from the network. The master files for the zones over which this server has secondary authority (`theirs.bak` and `theirs.rev.bak`) must contain back-up copies of the information about the associated zone, as received from the zone's primary server and written to the file by the local `named` process. This information must not be changed directly.

One server can serve as a master for multiple zones; it can be designated as primary for some and secondary for others. Queries are resolved by consulting the local cache (for authoritative and nonauthoritative data), any forwarding servers listed in the `/etc/named.boot` file, or root domain name servers on the Internet.

This method does not require (but can use) a remote forwarding server that can resolve recursive queries. It does require direct access to the root domain name servers on the Internet.

The decision to make a domain name server that is running on the local Cray host a master server depends on your local network's domain service configuration. You might prefer to let other servers perform most of the domain name resolution for the Cray host by running the Cray host's domain name server as a slave or caching-only server. In this way you can reserve the power of the Cray host for application processing. You might, on the other hand, prefer to give authority over certain domains to this server even though this may place slightly greater demands on the Cray host's resources.

2.2.8.3.5 Using the Domain Name Resolver Library Routines

Note: Use of the resolver client is not allowed with the Cray ML-Safe configuration of the UNICOS system. See Section 2.2.8.3, page 77, for more information about `named`(8) in UNICOS security.

The resolver library routines are the client portion of the domain name server. It resolves names whenever the `/etc/hosts.usenamed` file exists (see `gethost`(3) for exceptions). The optional `resolv.conf` file allows configuration of these routines.

The `/etc/resolv.conf` file can specify, on a line beginning with the keyword `domain`, the domain that contains the local host; otherwise, the resolver gets the default domain name from `gethostname`(2). On separate lines beginning with the keyword `nameserver`, the file must specify the Internet address of each remote host running domain name servers. These servers recursively process queries from the local Cray host. The following sample `/etc/resolv.conf` file specifies the local domain and two such remote hosts:

```
# Sample /etc/resolv.conf file

domain ourdomain.com
nameserver 127.0.0.1
nameserver 123.45.67.89
nameserver 234.56.78.90
```

The domain name resolver library routines do not cache any responses received (all queries are resolved independently of each other), which could result in slow responses and increased network traffic if a local name server daemon is not running. Therefore, it is recommended to run a local name server when the system is configured to resolve host names by using the Domain Name Service.

2.2.8.4 The `sendmail` Daemon

The `sendmail`(8) daemon performs Simple Mail Transfer Protocol (SMTP) operations.

> **Note:** For UNICOS security, the `/usr/mail` and `/usr/spool/mqueue` directories must be installed as multilevel directories. See *General UNICOS System Administration*, for information on installing multilevel directories with UNICOS security.

With the UNICOS system, it is not necessary to use the `newaliases`(1) command when implementing the SMTP. The program that implements the SMTP is called `sendmail`(8). The `sendmail` program implements both the client side and the server side of SMTP mail services; the mode of operation is determined by the manner in which `sendmail` is invoked. The `sendmail` program is in the `/usr/lib/sendmail` file, as it is on most other systems that run under operating systems based on 4.4 BSD. As a client, `sendmail` is not called directly by the user, but indirectly through the UNICOS `mail`(1) interface.

The `sendmail` daemon usually begins execution at system startup when it is invoked from the `/etc/config/daemons` shell script. If it is not started in this manner, the super user must invoke `sendmail`, as in the following example:

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

With the Cray ML-Safe configuration of the UNICOS system, `sendmail` must be invoked by a security administrator or a system administrator (active `secadm` or `sysadm` category). The following is an example of enabling security administrator status, invoking `sendmail`, and disabling security administrator status:

```
setucat secadm
/usr/lib/sendmail -bd -q30m
setucat 0
```

For a system administrator, use `setucat sysadm` instead of `setucat secadm`.

The `-bd` option specifies that `sendmail` must operate as an SMTP server process; the `-q` option sets the frequency with which `sendmail` must process mail in its queue. In this example, the `sendmail` daemon processes queued mail every 30 minutes. If you plan to run the `sendmail` daemon, add a line similar to the preceding command line to the `/etc/config/daemons` file. Set `-q` to the frequency you desire (see `sendmail`(8) for more information on how to specify this option).

The `/usr/lib/sendmail.cf` file, referred to as the `sendmail` configuration file, contains a set of directives to the `sendmail`(8) program. These directives instruct the program about how to interpret and deliver mail messages to and from the UNICOS system. The `sendmail` configuration file is distributed as a skeleton that you must customize for your site.

Warning: Modification of the `/usr/lib/sendmail.cf` file must comply with the guidelines set forth in the single-user mode descriptions in the "UNICOS security feature" section of *General UNICOS System Administration*. In addition, the `sendmail` daemon is running as a security administrator or system process when commands used in `/usr/lib/sendmail.cf` are executed. Therefore, additional care must be taken to ensure that the operations that these commands perform follow the rules and restrictions that are enforced for a security administrator, as described in the "UNICOS security feature" section of *General UNICOS System Administration*.

The `/usr/lib/sendmail.cf` file should not contain user-specified information unless that information has been verified to be acceptable by a security administrator. Users should be encouraged to use their own `.forward` files to customize their relationship with `sendmail` wherever possible.

None of the suggested modifications in this description violate the intent of this warning.

The lines that you must change are accompanied by comment lines that describe the changes to be made. At a minimum, you will need to make changes to the following lines:

- Near the beginning of the file, on the line reading

  ```
  DDdomain.domain
  ```

  change `domain.domain` to the domain name of your network.

  > **Note:** You can obtain the domain name of your network from the `domain` parameter of the `resolve.conf` file.

- If your system host name, as configured by the `hostname`(1) command (see Section 2.2.9.4, page 108), does not include your domain name, locate the following line near the beginning of the file:

  ```
  #DE$D
  ```

  Uncomment the line by removing the initial #.

- Near the beginning of the file, on the line reading

  ```
  Cw
  ```

  append the host name of the UNICOS system, all names from the `/etc/hosts` file or domain name server that refer to specific network interfaces on the UNICOS system, and any other aliases for the UNICOS system. Consider the following examples for Model E and GigaRing based systems:

  If Model E based UNICOS system is given the official host name `cray`, has two separate HYPERchannel interfaces labeled `cray-np0` and `cray-np1`, and also has been given the alias `supercomputer`, the line must be modified to read as follows:

  ```
  Cwcray cray-np0 cray-np1 supercomputer
  ```

  If a GigaRing based UNICOS system is given the official host name `cray`, has two separate FDDI interfaces labeled `cray-gfddi0` and `cray-gfddi1`, and also has been given the alias `supercomputer`, the line must be modified to read as follows:

  ```
  Cwcray cray-gfddi0 cray-gfddi1 supercomputer
  ```

  The actual order of the names on the line is not significant.

If you want the UNICOS system to interpret mail addresses and deliver mail messages directly, you must rewrite the `sendmail` configuration file for your specific needs. However, if you are integrating the UNICOS system into an existing network on which another system already serves as a central clearinghouse for mail traffic on the network, you can modify the distributed UNICOS `sendmail` configuration file as follows:

- Near the beginning of the file, on the line reading

  ```
  DMmailhost
  ```

  change `mailhost` to the host name of the central mail system.

After you have identified a central mail system on the network, you can modify the `sendmail` configuration file to have the UNICOS system send all or some of its mail to the central mail system, as follows:

- To have all nonlocal mail automatically forwarded to the central mail system, locate the following line near the beginning of the file:

  ```
  #DA$M
  ```

  Uncomment the line by removing the initial #.

- To have all mail forwarded to the central mail system, locate the following line near the beginning of the file:

  ```
  #DX$M
  ```

  Uncomment the line by removing the initial #. This disables local delivery; no mail can be delivered locally on the Cray system.

- To have mail with nonlocal addresses of the form *host* `!` *user* (using the UUCP style) sent to the central mail system, locate the following line near the beginning of the file:

  ```
  #DU$M
  ```

  Uncomment the line by removing the initial #.

- To have mail with addresses that specify nonlocal domains sent to the central mail system, locate the following line near the beginning of the file:

  ```
  #DN$M
  ```

  Uncomment the line by removing the initial #.

  Nonlocal domains are addresses of the form *user* @ *host.domain*; *domain* is not the same as the local domain that is specified in the configuration file.

For more information about `sendmail`, see the *Sendmail Installation and Operation Guide* by Eric Allman.

2.2.8.5 The snmpd Daemon

Note: The use of snmpd is not allowed with Cray ML-Safe configuration of the UNICOS system.

The snmpd daemon, also known as a *server/agent*, performs Simple Network Management Protocol (SNMP) operations on Cray systems. The agent resides in the background and listens for SNMP requests on port 161. When a request is received from a management station, snmpd performs the requested operations, as defined by RFC 1155 and 1157, and it provides management variables from the management information base (MIB), as defined by RFC 1213. See Appendix A, for a list of the MIB variables that Cray systems supports.

The following steps enable the snmpd daemon:

1. Add the SNMP and SNMP-trap ports to the /etc/services file by inserting the following lines (if they are not already there):

   ```
   snmp 161/udp
   snmp-trap 162/udp
   snmp-trap 162/tcp
   ```

2. Assign management station community names to the /etc/snmpd.conf file. This is a security feature in the SNMP protocol. Any community not listed in /etc/snmpd.conf is not serviced (see the following section for more details).

3. Ensure that /etc/snmpd is started by the /etc/config/daemons file when the system is initialized.

Cray systems support all operations defined by RFC 1157.

2.2.8.5.1 The /etc/snmpd.conf File

The /etc/snmpd.conf file contains a list of communities that are allowed to access the SNMP agent on the Cray system. The following is the sample /etc/snmpd.conf file that is in the source file. In this sample, the file contains keywords and parameters that specify access limits.

```
######################################################################
#
# Sample snmpd configuration file:
#
# 1. Fill-in the value for "sysContact" and "sysLocation" below, e.g.,
#
#       variable        sysContact      "joe operator <joeo@mw.cray.com>"
```

```
#
#       variable        sysLocation     "upstairs machine room"
#
#    All the other objects in the system group are automatically
#    filled-in by the agent.
#
# 2. If your site has a management station that listens for traps,
#    fill-in the information for the trap sink, e.g.,
#
#       trap            traps           a.b.c.d
#
#    where "traps" is the community that the traps should be logged
#    under and a.b.c.d is the IP-address of the host where a trap
#    sink is listening on UDP port 162.
#
# 3. Fill in community statements to provide the correct access to
#    the workstations running the manager programs.
#
#       community       name    address         access          view
#
#    where "name" is the name that is used to make snmp queries,
#    "address" is the ip address of the host that may use the
#    community, "access" is one of readOnly, readWrite, or none,
#    and "view" is the subid of the view applicable to this community
#    name.  An "address" of 0.0.0.0 allows all ip-addresses to use the
#    community.
#
# 4. Fill in view statements that define which portion of the mib a
#    particular community can access.
#
#       view            subid           mib_subtree
#
#       where subid is a mib sub-identfier that identifies this view.
#       Only the "n" portion need be changed, n can be any number from 1
#       to 255.  mib_subtree can be any combination of mib sub identifiers.
#
########################################################################

readall;

community       public          localhost               readOnly
community       manage          localhost               readWrite       1.17.1
community       system          localhost               readOnly        1.17.2
```

```
view            1.17.1          system interfaces ip
view            1.17.2          system

logging         file=snmpd.log  size=500
logging         slevel=fatal    slevel=exceptions       slevel=notice
logging         sflags=close    sflags=create           sflags=zero

variable        sysContact      "Your name <Your@address>"
variable        sysLocation     "Your location "

trap            traps           localhost
```

2.2.8.5.2 Using the Set Operation to Make Configuration Changes

The `snmpd` daemon supports the use of the set operation on the MIB variables, as defined by RFC 1213. You can use the `snmptest`(1) command to make configuration changes. Follow the prompts after you enter `snmptest`.

Note: When you use `snmp` set requests to make configuration changes, the changes remain in effect only until the next boot of the UNICOS system. These changes do not affect the configuration files or the install tool database.

You can make the following network configuration changes with `snmp` set requests:

- Add, delete, and change values of routes, as follows:
  - To add a new route, set the `ip.iproutetable.iproutentry.ipnexthop.128.162.3.4` variable to `128.162.6.7`, and set the `ip.iproutetable.iproutentry.iproutetype.128.162.3.4` variable to `3` if it is to a directly connected network or `4` if it is to an indirectly connected host or gateway. This set operation is equivalent to the `route add 128.162.3.4 128.162.6.7` command.
  - To delete a route, set the `ip.iproutetable.iproutentry.iproutetype.128.162.3.4` variable to `2` (this sets the type to `invalid`). This set operation is equivalent to the `route delete 128.162.3.4` command.
  - To change the values of an existing netmask, set the `ip.iproutetable.iproutentry.iproutenetmask.1.2.3.0`

variable to `ff ff ff 00`. This set operation is equivalent to the `route change 1.2.3.0 -netmask 0xffffff00` command.

- Change network interfaces, as follows:
  - To change the logical state of a network interface to `up`, set the `interfaces.iftable.ifentry.ifadminstatus.1` variable to `1`. For Model E based systems, this set operation is equivalent to the `ifconfig np0 up` command, where `np0` is the first interface. For GigaRing based systems, this set operation is equivalent to the `ifconfig gfddi0 up` command, where `gfddi0` is the first interface.
  - To change the logical state of network interface to `down`, set the `interfaces.iftable.ifentry.ifadminstatus.1` variable to `2`. For Model E based systems, this set operation is equivalent to the `ifconfig np0 down` command. For GigaRing based systems, this set operation is equivalent to the `ifconfig gfddi0 down` command.
  - (For Model E based systems only) To add a `hyroute` entry on interface `14` (HIPPI), set the

    ```
    mgmt.mib-2.at.atTable.atEntry.atPhysAddress.14.1.128.162.94.13
    ```

    variable to `01 00 00 07 00 20 00 30`. This set operation is equivalent to the `hyroute` command directive

    ```
    add direct 128.162.94.13 01000007 0020 0030 mtu
    ```

    where `128.162.94.13` is the Internet address of the host name, `01000007` is the I-field, `0020` is the minor input number, and `0030` is the minor output number. The mtu is the mtu of the interface as specified by the `ifconfig`(8) command.

    Following is an example of adding a `hyroute` entry on a HIPPI interface:

```
poplar09$ snmptest squall write
Please enter the variable name: $S
Request type is SET REQUEST
Please enter the variable name:
         mgmt.mib-2.at.atTable.atEntry.atPhysAddress.14.1.128.162.94.13
Please enter variable type [1|s|x|d|n|o|t|a]: o
Please enter new value: 01 00 00 07 00 20 00 30
Please enter the variable name:
Received GET RESPONSE from 128.162.82.6
requestid 0x4ef53739 errstat 0x0 errindex 0x0
```

```
Name: mgmt.mib-2.at.atTable.atEntry.atPhysAddress.14.1.12.8.162.94.13
OCTET STRING-  (hex): 01 00 00 07 00 20 00 30
Please enter the variable name: $Q
poplar09$
```

- (For Model E based systems only) To add a `hyroute` entry on interface `1` (HYPERchannel), set the

```
ip.ipNetToMediatable.ipNetToMediaentry.ipNetToMediaphysaddress.1.128.162.3.4
```

  MIB variable to `0 0 66 77`. This set operation is equivalent to the `add direct 128.162.3.4 6677 ctl access mtu` command, where `ctl` and `access` are the defaults for the interface. The mtu is the mtu of the interface as specified by the `ifconfig` command.

- (For Model E based systems only) To delete a `hyroute` entry on interface `1`, set the

```
ip.ipNetToMediatable.ipNetToMediaEntry.ipNetToMediatype.1.128.162.3.4
```

  variable to `2`.

• Turn IP forwarding on or off, as follows:

- To turn on IP forwarding, set the `ip.ipForwrding.0` variable to `1`. This set operation is equivalent to the `netvar -f on` command.
- To turn off IP forwarding, set the `ip.ipForwrding.0` variable to `2`. This set operation is equivalent to the `netvar -f off` command.

• Enable or disable the sending of SNMP authentication traps, as follows:

- To enable the sending of SNMP authentication traps, set the `snmp.snmpEnableAuthenTraps.0` variable to `1`.
- To disable the sending of SNMP authentication traps, set the `snmp.snmpEnableAuthenTraps.0` variable to `2`.

• Change the text describing the system contact and location, as follows:

- To change the text describing the system contact, set the `system.sysContact.0` variable to any ASCII string less than 255 characters.

  Following is an example of changing the `sysContact` MIB variable to the ASCII string `phil`:

```
poplar09$ snmptest
usage: snmptest gateway-name [community-name] [-P port]
poplar09$: snmptest squall write
Please enter the variable name: $S
Request type is SET REQUEST
Please enter the variable name: mgmt.mib-2.system.sysContact.0
Please enter variable type [i|s|x|d|n|o|t|a]: s
Please enter new value: phil
Please enter the variable name:
Received GET RESPONSE from 128.162.82.6
requestid 0x4ef53739 errstat 0x0 errindex 0x0
Name: mgmt.mib-2.system.sysContact.0
OCTET STRING- (ascii): phil
Please enter the variable name $Q
poplar09$
```

– To change the text describing the system location, set the `system.sysLocation.0` variable to any ASCII string less than 255 characters.

Following is the list of MIB variables that you can set. For a definition of data formats, see RFCs 1155 and 1213.

MIB variable	Setting
`sysLocation`	Can be set to any ASCII octet string.
`sysContact`	Can be set to any ASCII octet string.
`sysName`	Can be set to any ASCII octet string.
`ifAdminStatus`	Can be set to either `1` (up) or `2` (down).
`snmpEnableAuthenTraps`	Can be set to either `1` (enabled) or `2` (disabled).

`ipForwarding`

Can be set to either `1` (forwarding, acting as a gateway) or `2` (not forwarding, not acting as a gateway).

`ipDefaultTTL`

If set, it must be `255`, the default IP maximum time-to-live.

`ipRouteDest`

If set, it must be the IP address of the route's destination.

`ipRouteIfIndex`

If set, it must be an integer with the value of the interface index of the route.

`ipRouteMetric1`

Can be set to `-1` (no metric), `0` (a directly connected network, not a gateway), or `1` (a route to a gateway)

`ipRouteMetric2`

If set, it must be `-1` (no metric).

`ipRouteMetric3`

If set, it must be `-1` (no metric).

`ipRouteMetric4`

If set, it must be `-1` (no metric).

`ipRouteMetric5`

If set, it must be `-1` (no metric).

`ipRouteNextHop`

Can be set to the IP address of the gateway of the route to create a new route or change an existing route.

`ipRouteType`

Can be set to `2` (invalid, to delete the route), `3` (direct, to create a route to a direct network), or `4` (remote, to create a route to a gateway).

`ipRouteMask`

Can be set to a valid subnet mask if the destination is a network, or it can be set to `255.255.255.255` if the destination is a host.

`atphysaddress`

This is the hexadecimal octet string of the physical address. To delete the entry, provide a string of length 0; for HYPERchannel, provide a string of length 4; for FDDI or Ethernet, provide a string of length 6; for HIPPI, provide a string of length 8.

`atnetaddress` or `ipNetToMedianetaddress`

If either variable is set, it must be to the IP address of this entry.

`atifindex` or `ipNetToMediaifindex`

If either variable is set, it must be to the value of the interface index for this entry.

`ipNetToMediaphysaddress`

This is the hexadecimal octet string of the physical address. For HYPERchannel, provide a string of length 4; for FDDI or Ethernet, provide a string of length 6; for HIPPI, provide a string of length 8.

`ipNetToMediaType`

Can be set to `2` (invalid, an invalidated mapping), `3` (dynamic), or `4` (static, if this is to be a permanent Address Resolution Protocol entry).

2.2.8.6 The `ntpd` Daemon

The `ntpd` daemon provides the mechanisms Network Time Protocol (NTP) to synchronize time and coordinate time distribution in a large, diverse internetwork. The servers that implement this mechanism are known as *timeservers*. Timeservers synchronize local clocks within a subnet by propagating time information from primary hosts, which obtain national standardized time information from radio or other very accurate sources. These servers can also redistribute reference time by means of local algorithms and time daemons. A timeserver is either a server or a peer read from the NTP configuration file (see Section 2.2.8.6.1, page 95).

The NTP protocol is implemented on Cray systems by the `ntpd` daemon process. When `ntpd` is started from the `tcpstart`(8) script, it begins gathering statistics from its timeservers. It sends several messages to each candidate timeserver. Then each timeserver returns a reply message which contains several items, including the timeserver's local time and the claimed accuracy of its clock. By measuring the magnitude and distribution of the packet turnaround times for each timeserver, and by comparing the reported accuracy of each timeserver clock with the others, the `ntpd` daemon determines which of the candidate timeservers is currently the most accurate from the perspective of the local host. Because clock accuracy, turnaround-time magnitude, and turnaround-time distribution are all considered by the selection algorithm, there is no guarantee that the closest timeserver (smallest average turnaround time) is selected as the best timeserver.

To increase the possibility of making a good initial timeserver selection, `ntpd` performs several measurements during the time span of several minutes before actually making the selection. When the initial selection is made, `ntpd` might call `settimeofday`(2) to synchronize the time on the local host with that of the timeserver that it selected. When this happens, local users observe a *time warp*, a term used to describe an instant in which local time appears discontinuous. This initial time warp can be quite long. To prevent this time warp from affecting UNICOS applications, it strongly recommended that the `/etc/ntp -s -f` *host1 ... hostn* command be executed prior to invoking `ntpd` (*host1 ... hostn* are servers and peers taken from the `/etc/ntp.conf` file). Usually, this command forces the local time to be synchronized with one of the timeservers. This makes the initial time warp occur during system startup and, therefore, before the user processes begin. Alternatively, you can invoke the `/etc/ntpstart.sh` script. This script issues the appropriate `ntp` command before invoking `ntpd`.

Note: Time warp can affect software (for example, performance tests) that depends on smooth time and accounting. It can also affect the performance of such commands as `make`(1), `alarm`(2), and `sleep`(1).

After the local time is set, it usually remains accurate. However, `ntpd` continues to poll all configured timeservers at varying rates. The timeserver considered the most accurate is polled approximately once every 64 seconds; the timeservers initially considered less accurate are polled less often, perhaps once every 512 or 1024 seconds. After each poll, `ntpd` determines whether the local time is sufficiently accurate. If not, `ntpd` begins the process of determining whether the time difference is caused by clock drifting, and not by unexpected packet delays. Usually, it is packet delays that introduce the doubt about the time accuracy. However, when `ntpd` becomes convinced that its local clock has drifted too far from real time, it must adjust the local clock again. Usually, `ntpd` can use the `adjtime`(2) system call to gradually speed up or slow down the system clock. Occasionally, the `adjtime`(2) system call is not sufficient, and `ntpd` must call `settimeofday`(2). This creates another time warp. Fortunately, this is a relatively rare occurrence on Cray systems. Most of the time, `ntpd` can use the `adjtime`(2) system call to keep local time synchronized with network time. Unless something very unusual happens, such as simultaneously losing contact with most time servers, the system does not need to create time warps by using `settimeofday`(2). If this type of backward time movement is considered unacceptable at your site, `ntpd` must not be run on your server. In reality, however, it is not very probable that these adjustments will adversely affect UNICOS processes.

The `ntpd` daemon dynamically selects the best timeserver to use at any given time. If a path from the local host to its best timeserver becomes congested or lost, `ntpd` automatically selects another timeserver with which to synchronize. When the path later clears, `ntpd` can automatically revert to using the original timeserver if its performance is again superior to that of the other available timeservers.

The `ntpd` daemon is robust enough to ignore data from timeservers that appear suspicious. For example, `ntpd` disregards data from even a close timeserver if the time it reports disagrees substantially with the time reported by most other servers. Consequently, it is not likely to be fooled by a nonfunctional timeserver if a sufficient number of candidate timeservers are configured.

This implementation is based on the NTP version 1 protocol, described in detail in RFC 1059. Further information about NTP can be found in RFC 1119 and RFC 1129.

Note: The use of the `ntpd` daemon is not allowed with the Cray ML-Safe configuration of the UNICOS system.

2.2.8.6.1 The `/etc/ntp.conf` File

To enable the Network Time Protocol (NTP) facility, you must create the `/etc/ntp.conf` file, supplying timeserver information as shown in the following example:

```
# /etc/ntp.conf file

precision -10
server      wwvb.isi.edu
peer        umd1.umd.edu
server      130.126.174.40
```

`precision`	Specifies the accuracy of the local clock to the nearest power of 2 seconds. If you do not include this line, `ntpd` is forced to try to determine it; this process is not very effective. For Cray systems, a precision of `-10` gives good results, but your system's precision requirements can differ according to its frequency. Use the following

guidelines for setting the precision of your local clock:

Frequency	Precision
60 Hz	2**-6
100 Hz	2**-7
1000 Hz	2**-10

`server`
: In the first instance, `server` identifies a remote host with which the local host can synchronize. This remote host does not need to know anything about the local host, because the remote host does not synchronize with the local host under any circumstances. This `server` line consists of the keyword `server`, followed by at least one blank character, followed by the name of a remote host. The remote host's name can be specified in any format that is compatible with the local implementation of `gethostbyname`(3).

  In the second instance, `server` specifies the server's address in the familiar dot notation.

`peer`
: Like the `server` line, this line identifies a remote host with which the local host can synchronize. However, the `peer` line also implies that the remote host can synchronize with the local host, if appropriate. For this peer relationship to work properly, if host A is a peer in host B's configuration file, host B must be a peer in host A's configuration file. Configuring a remote host as a peer without the consent of the remote host is functionally equivalent to configuring the remote host as a server.

  The `peer` line consists of the keyword `peer`, followed by at least one blank character, followed by a host name.

  Note: The `ntp`(8) man page, and this section, present specific requirements for correctly selecting NTP peers and servers. Refer to this documentation before you select your peers.

`passive`	(Not shown in the example). This identifies a type of relationship with a remote host that is supported, but not recommended. The configuration syntax is like that of `server` and `peer`. The passive relationship causes packets to be sent to the specified remote host when it polls the local host. This prevents the local host from initiating synchronization with the remote host, but it forces the local host to synchronize with the remote host through remote demand. Because this relationship gives too much control to the remote host, and it violates the traditional client/server model, Cray does not recommend configuring any passive hosts in the configuration file.

2.2.8.6.2 NTP Server Categories

The *strata*, based on their distance from NTP servers that have reference clocks NTP servers are divided into categories called physically attached. Servers deriving their time from attached clocks are defined as being in stratum 1. Servers that synchronize with stratum-1 servers are in stratum 2, servers that synchronize with stratum-2 servers are in stratum 3, and so on. This classification scheme creates a hierarchy of time servers, with those in the lower numbered strata being more accurate and reliable.

If you are configuring only the Cray system to run NTP, the configuration task is relatively simple. Configure the `/etc/ntp.conf` file with three or four Internet stratum 1 and 2 servers; this is a good trade-off between the benefits of redundancy and the costs of protocol processing.

Configuration becomes more complex when you are trying to design the overall NTP architecture of a local subnet. Consider the example in which you want to begin running NTP not only on your Cray system, but also on several other local hosts (if their NTP daemon executable files are available to you). You must determine which servers will be in stratum 2 and which servers will be in stratum 3. A good guideline is to select three of your systems with the best clocks and your fileservers to be local servers. Choose six stratum-1 Internet hosts and pair them off (a list of NTP servers is available through anonymous FTP from `louie.udel.edu` in the `pub/ntp/clock.txt`) file. For each pair of stratum-1 hosts, choose an external stratum-2 host that synchronizes with primary servers other than that pair. Then configure each of your local servers in the following manner:

`server` *one_of_the_stratum_1_hosts*

```
server    the_other_stratum_1_host
server    corresponding_external_stratum_2_host
peer      one_of_the_other_local_servers
peer      the_third_local_server
```

This is an extremely robust configuration. It is resistant to nonfunctional timeservers and generates less traffic than would occur if multiple machines were all communicating with the same set of external peers. The second stratum-1 host in each pair can be further away, because there is usually less traffic.

The remaining machines can now synchronize with your three local stratum-2 servers. They can be configured with the following entries:

```
server    local_stratum_2
server    another_local_stratum_2
server    the_third_local_stratum_2
```

2.2.8.7 The `inetd` Daemon

The `inetd` command provides a mechanism for starting daemons that are listed in the `/etc/inetd.conf` file. The `inetd` daemon is a *super server* because it listens for incoming requests for the services. Whenever a request is received, `inetd` starts the individual server process. Following is a list of the daemons that `inetd` starts and the service each performs.

Note: The daemons listed with asterisks perform services that must not be configured with the Cray ML-Safe configuration of the UNICOS system.

Daemon	Service
`fingerd`(8)	Remote user information display server
`ftpd`(8)	Internet file transfer protocol server
`ntalkd`(8)*	Visual communication server
`rexecd`(8)	`rexec`(3) library routine server
`rlogind`(8)	Remote login server
`rshd`(8)	Remote shell server
`telnetd`(8)	Internet virtual terminal server
`tftpd`(8)*	Internet trivial file transfer protocol server
`ypupdated`(8)*	Network information service (NIS) update server
`rstatd`(8)*	Remote status daemon
`rusersd`(8)*	Remote user information daemon

`sprayd`(8)*	Remote spray daemon (used for testing)
`rwalld`(8)*	Remote write all daemon

If you are using the UNICOS ICMS for your configuration, you can use the `Configure System -> Network configuration -> TCP/IP configuration -> Internet daemon` menu to configure the list of services for which `inetd` listens. The following shows the sample `/etc/inetd.conf` file that is included in the source file. Table 3, page 100, describes each column as it relates to creating or modifying this file.

```
#
# Sample /etc/inetd.conf file
# Internet server configuration database
#
ftp     stream  tcp     nowait  root    /etc/ftpd     ftpd
telnet  stream  tcp     nowait  root    /etc/telnetd  telnetd
5100    stream  tcp     nowait  root    /etc/telnetd  telnetd -D report
shell   stream  tcp     nowait  root    /etc/rshd     rshd
login   stream  tcp     nowait  root    /etc/rlogind  rlogind
exec    stream  tcp     nowait  root    /etc/rexecd   rexecd
# Run as user "uucp" if you don’t want uucpd’s wtmp entries.
#uucp   stream  tcp     nowait  root    /etc/uucpd    uucpd
finger  stream  tcp     nowait  nobody  /etc/fingerd  fingerd
#tftp   dgram   udp     wait    tftp    /etc/tftpd    tftpd
#comsat dgram   udp     wait    root    /etc/comsat   comsat
#ntalk  dgram   udp     wait    root    /etc/ntalkd   ntalkd
echo    stream  tcp     nowait  root    internal
discard stream  tcp     nowait  root    internal
chargen stream  tcp     nowait  root    internal
daytime stream  tcp     nowait  root    internal
time    stream  tcp     nowait  root    internal
echo    dgram   udp     wait    root    internal
discard dgram   udp     wait    root    internal
chargen dgram   udp     wait    root    internal
daytime dgram   udp     wait    root    internal
time    dgram   udp     wait    root    internal
#
#
# RPC services syntax:
# <rpc_prog>/<vers> <socket_type> rpc/<proto> <flags> <user> <pathname> <args>
ypupdated/1   stream  rpc/tcp wait root /etc/ypupdated ypupdated
rstatd/2-4    dgram   rpc/udp wait root /etc/rstatd    rstatd
rusersd/1-2   dgram   rpc/udp wait root /etc/rusersd   rusersd
```

```
sprayd/1      dgram   rpc/udp wait root /etc/sprayd    sprayd
rwalld/1      dgram   rpc/udp wait root /etc/walld     rwalld
```

Table 3. `/etc/inetd.conf` file columns

Column number	Column name	Description
1	Service name	Specifies the service. If a service is not needed, you can disable it by commenting it out with a # preceding the line. In the preceding sample file, `uucp`, `tftp`, `comsat`, and `ntalk` services have been disabled. The `inetd` program searches the `/etc/services` file for the specified service and obtains the port number, which can also be directly specified in this column, in place of the service name. The `inetd` daemon opens the port number and begins listening for incoming requests on it. For RPC-based servers, the server name is followed by a slash (/), which is followed by the version number of the protocol (for example, `sprayd` /1 indicates version 1 of `sprayd`, and `rstatd` /2-4 indicates versions 2 through 4 of `rstatd`).
2	Socket type	Specifies the use of either a stream (TCP) or a datagram (UDP) socket.
3	Protocol type	Specifies which protocol is used: TCP (socket type is `stream`), or UDP (socket type is `dgram`). For RPC-based servers, the protocol type is preceded by `rpc/` (for example, `rpc/tcp` or `rpc/udp`).
4	Single-threaded / multithreaded	Indicates whether a new server's status is single-threaded (`wait`) or multithreaded (`nowait`) (see `inetd.conf`(5) for more information).
5	User	Describes the access permissions for the daemons. The `ftp`, `telnet`, `shell`, and `login` servers require root permission; the `finger` and `tftp` servers should be run as users with limited capability. In the preceding sample file, the `finger` server runs with the capability of user `nobody`; the `tftp` server (if enabled) runs with the capability of user `tftp`. You must create these users and assign limited permissions to them.

Column number	Column name	Description
6	Server program	Indicates the location of the program. The `inetd` server requires an entry here. For internal services, put `internal` in this column. For example, in the preceding sample file, the `telnet` daemon is located in `/etc`; `echo` is an internal service.
7	Server program and arguments	Indicates the program's name and describes the arguments to be used on the `exec`(2) system call that `inetd` executes. In the preceding sample file, a `telnet` daemon is available on port 5100, which operates in diagnostics reporting mode. The `-D` argument is specified in the last column. See `telnetd`(8) for more information on this argument.

Note: For security reasons, it is recommends that you disable `tftpd`(8). If the site requires `tftdpd`, see Section 2.2.8.7.7, page 104, for configuration information. `tftpd` must not be run with the Cray ML-Safe configuration of the UNICOS system.

2.2.8.7.1 The `fingerd` Server

The server program for the remote `finger`(1B) routine is `fingerd`(8).

Note: The `fingerd` server must not be run with the Cray ML-Safe configuration of the UNICOS system.

2.2.8.7.2 The `ftpd` Server

The server program for the Internet file transfer protocol is `ftpd`(8). The `/etc/ftpd` command accepts the following options:

Option	Description
`-d`	Logs debugging information to the `syslogd` file.
`-l`	Logs each `ftp` session to the `syslogd` file.
`-t` *timeout*	Terminates an inactive session after *timeout* seconds
`-v`	Logs debugging information to the `syslogd` file.

See `ftpd`(8) for a complete list of options.

If users create their own $HOME/.netrc files in the working directories of the client hosts, they do not have to provide the ftp server with their user names or passwords. For information on the $HOME/.netrc files, see the *TCP/IP Network User’s Guide*. A version of ftpd that supports Kerberos authentication is available (see the *Kerberos Administrator’s Guide*).

If the user name is anonymous or ftp, and an ftp account is present in the password file, the user is allowed to log in by specifying any password. The chroot(2) system call is executed to change to the login directory of the ftp account. Because anyone can log in under anonymous, it is advisable to restrict the access privileges to this account: files and directories should not be owned by the ftp account, and read-only permission should be set. Because of the chroot system call, the ftp account must have a bin directory with the ls(1) and pwd(1) commands in it so that the ftp dir command works.

The /etc/ftpusers file contains the names of users who are denied access to ftp from a remote host. Each time an ftp user attempts to log in, the ftpd(8) server searches this file for the user’s login name. If the name is found, the user is denied access. Valid user names (that is, names of users who should have access) must not appear in the /etc/ftpusers file; if they do, access is denied them.

An example of an /etc/ftpusers file is as follows:

```
# Sample etc/ftpusers file

# denied ftp users
todd
doug
cheryl
bonnie
```

2.2.8.7.3 The ntalkd Server

The ntalkd program is the remote server used by the UNIX talk(1B) command. This program is a visual communication program that copies lines from your terminal to the terminal of another user.

Note: The ntalkd server must not be run with the Cray ML-Safe configuration of the UNICOS system.

2.2.8.7.4 The rexecd Server

The server program for the remote execution routine is rexecd, which provides an existing server for custom-designed network programs. (See the rexec(3) man page for more information.) For information about how users on hosts that run under operating systems based on UNIX can use remote execution for their own applications, see the`rexecd`(8)man page.

Remote users who want to use the `rexecd` server on the Cray system (the local host) must authenticate themselves with user names and passwords for their accounts. Users have three options for specifying names and passwords:

- Code the name or password into the local program.
- Direct `rexec`(3) to prompt the user for the name or password in the client program.
- Store the name or password in the user's `$HOME/.netrc` file on the client.

2.2.8.7.5 The `rlogind` and `rshd` Servers

The `rlogin`(1B) program allows remote login to other hosts that run under operating systems based on UNIX (see the *TCP/IP Network User's Guide.* The `rsh`(1) program allows the execution of one command on other hosts that run under operating systems based on UNIX, and it handles requests for the `rcp`(1) (remote copy) commands. Versions of `rlogind`(8) and `rshd`(8) that support Kerberos authentication are available (see the *Kerberos Administrator's Guide*, for details).

These programs allow users at other hosts to log in and execute commands on a Cray system.

The `rlogin`, `rcp`, and `rsh` programs provide the following:

- Automatic remote login. Users are not required to enter their user names and passwords on the serving hosts because the system verifies login information based on the `/etc/hosts.equiv`, the `$HOME/.rhosts`, and the `/etc/password` files.
- Terminal characteristics automatically passed to the serving host. Users are not required to specify their terminal type to the serving host.

To allow automatic user login, you can create an `/etc/hosts.equiv` file on the server host (see Section 2.2.5.3, page 37), or users can create their own `$HOME/.rhosts` files (see Section 2.2.11.2, page 121). One of these files must be properly configured to use the `rsh` and `rcp` commands. However, the `rlogin` command does not require these files. If they are not present, `rlogind` validates the user who is making the request by doing the following:

1. Prompts users for their user names on the server host.

2. Verifies that the user names are in the server host's password file and are password-protected; they must not have a null password.
3. Prompts users for their passwords, which they must supply before remote login operations can be performed.

2.2.8.7.6 The `telnetd` Server

The `telnetd`(8) program is a server to the DARPA standard `telnet` virtual terminal protocol. It operates by allocating a pseudo-terminal device for a client who is attempting to connect. After this allocation is complete, a login process is initiated on the slave side of the pseudo-terminal with `stdin`, `stdout`, and `stderr` assigned to the pseudo-terminal. `telnetd` manipulates the master side of the pseudo-terminal and uses the `telnet` protocol to pass data between the client and the login process.

The `telnetd` daemon is started by the `inetd`(8) daemon whenever a request comes in on the specified port. `telnetd` supports the following options, which can be specified in column 7 of the `/etc/inetd.conf` file as the program arguments (see `telnetd`(8) for a complete list of options):

`-r` *low_pty* *high_pty*

Specifies the incrementing range for finding a free pseudo-terminal. When allocating pseudo-terminals, `telnetd` begins at *low_pty* and continues incrementing by 1 until either a free pseudo-terminal is found, or *high_pty* has been tried. The defaults for *low_pty* and *high_pty* are 0 and 100 (the value of the `sysconf` variable `SC_CRAY_NPTY`), respectively.

`-h` Suppresses system message display. When a user specifies `telnet` to access a Cray system, a system message is displayed prior to the login prompt if this option is not specified. This system message indicates the remote host's name, operating system, and so on.

2.2.8.7.7 The `tftpd` Server

The server program for the Internet trivial file transfer protocol is `tftpd`, which does not have a login facility. Because it does not validate users, this server can be used to transfer only publicly readable files; however, the `tftpd` server supports Kerberos authentication. Thus, authentication of the client can be performed through the Kerberos subsystem; users can access all files that would be available to them if they were logged in locally.

For security reasons, you should disable the `tftp` service. However, if this service is needed, the following steps can be taken:

1. Create a user `tftp` file with limited privileges.

2. Modify the `/etc/inetd.conf` file to set the `tftp` access permission parameter to the user created in step 1 (column 5 of the `tftp` line in the `/etc/inetd.conf` file).

3. Edit the `/etc.tftpd.conf` file to include the directories you select if you want the inbound `tftp` to access only certain directories (such as `/tmp`, `/usr/tmp`, and so on). The `/etc/tftpd.conf` file contains a list of the directories that are available through `tftp` access. A directory not listed in this file is not accessible through `tftpd`.

You must provide the full path name of each directory specified in the `/etc/tftpd.conf` file (for example, `/tmp`, `/usr/tmp`, and so on) and its access permissions (`R` or `r` for read only, `W` or `w` for write only, and `RW`, `WR`, or `rw` for read / write). If the `tftpd` access permissions are different from the directory's usual access permissions, the directory's usual permissions override the `tftpd` access permissions. For example, if a directory cannot be read by others, putting that directory in the `/etc/tftpd.conf` file with read permission does not permit read access.

The following example shows a sample `/etc/tftpd.conf` file. In this example, `/garbage` is readable and writable through `tftp`, and `/tmp` is readable through `tftp`, if these directories set public read / write and read permissions, respectively.

```
#This is the tftpd configuration file.  It contains the list of
#directories and the type of access for these directories for tftpd.

#The format is as follows:
#       Mode    Directory
#       where Mode is either R (read only access via tftpd), W
#       (write only access via tftpd), or RW or WR (read/write
#       access via tftpd.)  Each directory has to be specified on
#       a separate line.
#
#
#NOTE tftpd is a big security risk and should not be used unless
#      necessary.
```

```
#
     RW  /garbage
     R   /tmp
```

Note: The `tftpd` daemon must not be configured with the Cray ML-Safe configuration of the UNICOS system.

2.2.9 Performing Startup Procedures

When the system starts up, the networking software is started by the `/etc/tcpstart` script, which is typically called by the network start-up script, `/etc/netstart` (which is, in turn, typically called by the system start-up script, `/etc/rc`).

Note: The `/etc/tcpstart` script that is supplied with the UNICOS system should not be modified directly.

The default `/etc/tcpstart` script supplied with UNICOS is designed to be sufficiently general in its capabilities to support the startup of networking on a majority of UNICOS systems (if the underlying configuration files explained in this section are configured correctly).

You can start the UNICOS networking software by using a method other than the supplied `/etc/tcpstart` file. Sometimes administrative procedures for starting networks are written already, or (infrequently) the local configuration has special needs that are too complex for the supplied `/etc/tcpstart` file. (If the latter is true, inform us of your needs so that they can be considered for future revisions of `/etc/tcpstart`.) Nevertheless, becoming familiar with the actions taken by the supplied `/etc/tcpstart` file is valuable, because any alternative script or method you use to start the UNICOS networking software must accomplish the goals described in this section.

The default for UNICOS is for the `/etc/tcpstart` script to perform the following functions:

1. Calls a local script (`/etc/tcpstart.pre`), if present
2. Initializes kernel networking variables
3. Updates the binary copy of the `/etc/hosts` database
4. Configures the host name of the system
5. With UNICOS security, loads the NAL, WAL, and IPSO maps
6. Initializes the networking interfaces

7. Calls a local script (`tcpstart.mid`), if present
8. Sets up routing
9. Starts the networking daemons related to TCP/IP
10. Calls a local script (`/etc/tcpstart.pst`), if present

Each of these steps is discussed in the following sections.

2.2.9.1 Calling the First Local Script

The `/etc/tcpstart` script begins by calling the local script `/etc/tcpstart.pre` if it exists and is executable. You should place the following in this script:

- TCP/IP start-up processes that are specific to your local system
- TCP/IP start-up processes that must be performed before the networking software is started
- TCP/IP start-up processes that cannot be accomplished by any of the other mechanisms provided

2.2.9.2 Initializing Kernel Networking Variables

To initialize kernel networking variables that are configurable while the system is running, the `/etc/tcpstart` script calls the `/etc/netvar` utility with the contents of the `/etc/config/netvar.conf` file as arguments to `netvar`. Lines in the `/etc/config/netvar.conf` file that begin with a # are considered comments and are ignored. For example, the following `/etc/config/netvar.conf` file turns on IP forwarding in the kernel (`-f on`), turns off sending of Internet control message protocol (ICMP) redirect packets (`-r off`), and sets the default TCP/IP send space to 32,768 bytes:

```
# /etc/config/netvar.conf file

# set networking kernel variable at boot time

-f on
-r off
-t 32768
```

See `netvar`(8) for a full list of the kernel variables it can set and the associated flags.

2.2.9.3 Updating the Binary Hosts Database

To ensure that the binary host name and address database `/etc/hosts.bin` is current with the information in the text file version `/etc/hosts`, the `/etc/tcpstart` script calls the `mkbinhost`(8) command. This command updates `/etc/hosts.bin` whenever `/etc/hosts` has changed, to avoid problems that could arise from incorrect name translations.

2.2.9.4 Configuring the Host Name

The `/etc/tcpstart` script next configures the running system with the system host name. Because a system can determine its host name in various ways, the `/etc/tcpstart` script follows this procedure:

1. If the `/etc/config/hostname.txt` file exists, the `/etc/tcpstart` script sets the system host name to the contents of the file.
2. Otherwise, if the `/etc/net/makehostname` script exists and is executable, the `/etc/tcpstart` script executes the script and sets the system host name to the output of its execution.
3. Otherwise, the `/etc/tcpstart` script sets the system host name to the system name compiled into the UNICOS kernel, as reported by executing the following:

   ```
   uname -n
   ```

   (See `uname`(1) for details.)

The rules to set the host name can be summarized, as follows:

- If you want the system host name to remain the same as the UNICOS kernel system name, do not do anything.
- If you want the system host name to be different, and the name you want is a simple character string, place it in the `/etc/config/hostname.txt` file. For example, if you want your system host name to be `mycray`, place the following in the `/etc/config/hostname.txt` file:

  ```
  mycray
  ```

- Alternatively, you can generate the system host name at boot time by creating a script to do so, placing it in the `/etc/config/makehostname` file, and making the script executable. For example, if you want to append the domain name `our.domain` to the kernel system name as reported by `uname`(1), place the following in the `/etc/config/makehostname` file:

```
# tiny script to create our host +
# domain name from our uname

echo ‘uname -n‘.our.domain
```

Then make the script executable by typing the following command:

```
chmod +x /etc/config/makehostname
```

2.2.9.5 Loading the Maps

With UNICOS security, you must load the NAL, WAL, and IPSO maps from the `/etc/config/spnet.conf` file. See `spnet`(8)) for a description of the `spnet.conf` file.

2.2.9.6 Initializing the Network Interfaces

After the host name is configured, the TCP/IP startup procedure should initialize the networking interfaces that permit TCP/IP communication over the attached networks.

2.2.9.6.1 Using the UNICOS `tcpstart` Script to Initialize Interfaces

The `/etc/tcpstart` script calls the utility script `/etc/initif` to initialize the networking interfaces. `/etc/initif` consults the `/etc/config/interfaces` file for information about each interface to be initialized.

If you are using the UNICOS ICMS for configuration of your networking interfaces, the UNICOS ICMS maintains the `/etc/config/interfaces` file with the information you supply by using the `Configure System -> Network configuration -> General network configuration -> Network interface configuration` menu.

If you are not using the UNICOS ICMS, you must place the information about your network interfaces directly in `/etc/config/interfaces`. Each line in `/etc/config/interfaces` describes the startup of one interface. The format of each line is as follows:

interface_name hycf_file `or` *arp_file address_family host destination*
[*ifconfig_arguments*]

hycf_file (Model E based systems only)	Name of a file containing the hardware address information associated with this interface. If *hycf_file* begins with a leading `/` (slash), it is assumed to be a full path name; otherwise, it is assumed to be the name of a file relative to the configuration directory `/etc/config`.
arp_file (GigaRing based systems only)	Name of a file containing the hardware address information associated with ghippi or gr interface only. If *arp_file* begins with a leading `/` (slash), it is assumed to be a full path name; otherwise, it is assumed to be the name of a file relative to the configuration directory `/etc/config`.
interface_name	Kernel identifier of the interface to be configured (for example, `lo0`, `np1`, and so on).
address_family	The address family of the address specified as the *host* argument. For Internet addresses, this is `inet`.
host	Internet address (or host name corresponding to the address, as found in the `/etc/hosts` database) to be associated with this interface.
destination	Internet address (or host name corresponding to the address, as found in the `/etc/hosts` database) of the destination for a point-to-point link. If this interface is not a point-to-point link, the column should contain `-` (one hyphen).
ifconfig_arguments	List of optional arguments for this interface that are passed to the interface configuration program `ifconfig`. This list consists of keywords followed by optional arguments (for example, `netmask 0xffffff00`, `-trailers`). See `ifconfig`(8)) for a full list of permitted arguments.

Note: If you are running with UNICOS security, also see Section 2.6, page 211, for security-related parameters.

For maximum performance between two Cray systems, an mtu of 65536 is recommended for HIPPI interfaces, as well as for Host-to-Host GigaRing interfaces.

For Model E based systems, you must ensure that the interface mtu is at least as large as the mtu specified in the `hycf` file. The mtu in the `hycf` file is the size of the largest IP datagram that can be sent to a given host. The interface mtu is the size of the largest IP datagram that can be received on this interface.

The following is a sample `/etc/config/interfaces` file for a Model E system that specifies three interfaces (and also the local loopback interface `lo0`):

```
#
# Configuration file for interfaces known to /etc/initif.
#
# File format is:
#
# name  hycf_file        family address     pt-to-pt-dest       args:
#                                                               netmask
#                                                               iftype
#                                                               broadcast
#                                                               mtu
#                                                               rbuf
#                                                               wbuf
#                                                               bg
#       np0 setup temporarily for 4801 through vme
#
lo0     -               inet    localhost       -
np0     /etc/hycf.ows   inet    hot-030         -         netmask 0xffffff00
                                                          iftype vme
np2     /etc/hycf.np2   inet    hot-fddi        -         netmask 0xffffff00
                                                          iftype n130
hi0     /etc/hycf.hippi inet    hot-hippi       -         netmask 0xffffff00
                                                          mtu 65536
```

The following is a sample `/etc/config/interfaces` file for a Cray J90 system that specifies three interfaces (and also the local loop-back interface `lo0`):

```
#
# Configuration file for interfaces known to /etc/initif (to be brought
# up at system startup by /etc/tcpstart).
#
# File format is:
#
# name  hycf_file       family  address        pt-to-pt-dest    args:
#                                                               netmask
#                                                               iftype
#                                                               broadcast
#                                                               mtu
#                                                               rbuf
#                                                           wbuf
#                                                           bg
#
lo0     -               inet    localhost       -
en0     -               inet    sn5194-ccn      -           netmask 0xffffff00
fddi0   -               inet    sn5194-fddi     -           netmask 0xffffff00
hi0     /etc/hycf.hippi inet    sn5194-hippi1  frost-hippi1  netmask 0xffffff00
                                                            mtu 65536
```

The following is a sample `/etc/config/interfaces` file for a GigaRing based system that specifies one active interface (and the active local loopback interface `lo0`):

```
#
# Configuration file for GigaRing network interfaces to be brought
# up by interfaces /etc/tcpstart through /etc/initif.
#
# Supported network interfaces are:
#
# name    arp_file         family address      pt-to-pt-dest       ifconfig_args
# ----------------------------------------------------------------------------
# gether0 -                inet   snXXXX-ether -              netmask 0xffffff00
# gfddi0  -                inet   snXXXX-fddi  -              netmask 0xffffff00
# ghippi0 /etc/ghippi0.arp inet   snXXXX-hippi -rbuf32 wbuf32 netmask 0xffffff00 hwloop
# gatm0   -                inet   snXXXX-atm   -              netmask 0xffffff00
# gr0     /etc/gr0.arp     inet   snXXXX-gr    -              netmask 0xffffff00
#
# name    arp_file         family address      pt-to-pt-dest      ifconfig_args
#
lo0     -                  inet   localhost    -
gether0 -                  inet   snXXXX-ether -              netmask 0xffffff00
```

2.2.9.6.2 Using Your Own Procedures to Initialize Interfaces

If you are not using the `/etc/tcpstart` script that is supplied with the UNICOS system, your startup procedure must initialize the networking interfaces for your attached networks.

You can initialize your system's interfaces by using the same `initif`(8) script that is used by the `/etc/tcpstart` script that is supplied by the UNICOS system. If you choose this method, consult the preceding section or `initif`(8) for operating details.

If you prefer not to use the `initif` script, you must arrange for your startup procedure to execute the following commands for each interface to be initialized at startup:

- `fddiload`(8) (IOS–E based systems that contain one or more FDDI interfaces only).

  Cray systems that have one or more FDDI interfaces must first have each of the FDDI channel adapters (FCAs) downloaded with FDDI microcode. The `fddiload` command performs this function. The microcode binary file resides in the `/etc/micro/fca1.ucode` file.

- `hyroute`(8) (Model E and model V based systems only).

  After the IOS channel is initialized for the networking interface (if appropriate), the interface must be initialized with information about the hardware addresses of the systems on its attached network. This is accomplished with the `hyroute`(8) command and the appropriate `hycf` file for the interface.

  For example, if the information about the hardware addresses of the systems attached to the network of interface `np0` is stored in the `/etc/hycf.np0` file, you must arrange for your startup procedure to perform the following command before proceeding with configuration of the `np0` interface:

  ```
  hyroute np0 -s /etc/hycf.np0
  ```

  For information about creating the proper `hycf` file for an interface, see Section 2.2.6.4, page 44.

- `ifconfig`(8)

  When you have initialized the hardware addresses for a network interface, you must initialize the interface to an active state by using the `ifconfig`(8) command. The `ifconfig` command initializes various parameters associated with the interface, including the specific Internet address that identifies the interface to the TCP/IP software.

  The following examples show a typical `ifconfig` command executed during system startup:

  For Model E based systems:

  ```
  ifconfig np0 mycray-net1 netmask 0xffffff00 iftype np
  ```

  For GigaRing based systems:

  ```
  ifconfig gfddi0 mycray-net1 netmask 0xffffff00 iftype np
  ```

In these examples, `mycray-net1` is the host name for the Internet address that is associated with the `np0/gfddi0` interface. The command also specifies that addresses for the hosts on this network have a subnet mask of `0xffffff00` (for more information about subnets and subnet masks, see Section 2.1.2.2, page 10).

The following examples show a typical `ifconfig` command for a HIPPI interface:

For Model E based systems:

```
ifconfig hi0 sn2402-hippi mtu 65536 netmask 255.255.255.0
```

For GigaRing based systems:

```
ifconfig ghippi0 sn2402-hippi mtu 65536 netmask 255.255.255.0
```

If you specify the `ptp` argument and the interface is brought up on the dedicated path, all HIPPI packets are sent in hold-connection mode. This means that a HIPPI connection is made the first time a packet is sent. The connection remains in place until the interface is configured down. Following is an example of the `ifconfig` command for a HIPPI point-to-point connection:

```
ifconfig hi0 sn2402-hippi sn5194-hippi ptp mtu 65536 netmask 255.255.255.0
```

The typical configuration for a Host-to-Host GigaRing interface is as follows:

```
ifconfig gr0 sn9132-gr netmask 0xffffff00
```

2.2.9.7 Calling the Midpoint Local Script

The `/etc/tcpstart` script next calls the local script `/etc/tcpstart.mid` if it exists and is executable. You should place the following in this script:

- Processes that are specific to your local system
- Processes that must be performed after the network interfaces have been initialized but before additional static routes have been installed
- Processes that cannot be accomplished by any of the other mechanisms provided

2.2.9.8 Setting up Routing

After the networking interfaces are initialized, the TCP/IP startup procedure should initialize the kernel tables for routing of network traffic. This initialization must be performed before the startup procedure starts any other networking software (such as system daemons), which may need routing information to communicate with remote hosts.

2.2.9.8.1 Using the UNICOS `tcpstart` Script to Set up Routing

The `/etc/tcpstart` script initializes the kernel tables for routing of network traffic from information placed in the `/etc/gated.conf` file. You can create the `gated.conf` file by using the `Configure System -> Network configuration -> TCP/IP configuration -> Routing` menu.

Usually, when the `gated`(8) routing daemon is started (see Section 2.2.8.1, page 61), it installs dynamic routing information from the `/etc/gated.conf` file to initialize the routing table in the kernel. However, if the `/etc/sdaemon` program indicates that the `gated` daemon is not starting as part of the group of TCP/IP daemons that start during system startup, the `/etc/tcpstart` script executes the `/etc/staticrts` script to initialize the kernel routing table with the static routes listed in the `/etc/gated.conf` file.

> **Note:** Even if you are not using dynamic routing (that is, the `gated`(8) program), the `/etc/staticrts` script assumes that the information that describes your static routes is located in the `/etc/gated.conf` file.

Using the `gated` routing daemon is the generally preferred method of installing routing information because of the dynamic routing capabilities of `gated`. For a full discussion of its capabilities and configuration, see Section 2.2.8.1, page 61. However, sites with simple networks that do not need the dynamic routing capabilities of `gated` can avoid the overhead of the daemon by disabling the `gated` daemon and relying on the `/etc/staticrts` script.

UNICOS extensions to the format of the `/etc/gated.conf` file allow configuration of all Cray proprietary routing features, such as configuring an mtu size for each route and the ability to impose route restrictions on users in a specific list of groups. Thus, it should always be possible to configure your system routing tables adequately by using the `/etc/gated.conf` file and either the `gated` daemon or the `/etc/staticrts` script.

If, however, it becomes necessary to configure your system startup procedures to use the `route` command directly, you can use the following methods to arrange for the `/etc/tcpstart` script to execute the appropriate `route` commands at system startup (in decreasing order of preference):

- Place a line in the `/etc/config/daemons` file to execute a local start-up script that contains the necessary `route` commands. See Section 2.2.9.9, page 116, for a description of this file. The line must be part of the TCP group of daemons, and it must be placed in the file before any other lines that belong to the TCP group of daemons. The line can have any string you want in the `tag` column; it must contain the string `YES` in the `start` column; and it must have a hyphen in the `kill` column, followed by the name of the script.

For example, if the script to be executed is `/etc/myroutes`, you must add the following line to the `/etc/config/daemons` file:

```
TCP    routes    YES    -    /etc/myroutes
```

- Replace the `/etc/staticrts` script that is supplied with the UNICOS system with a local start-up script that contains the necessary `route` commands.
- Edit the `/etc/tcpstart` file to add the necessary `route` commands before the call to `etc/sdaemon`, which starts the TCP group of daemons.

Caution: The last two methods of setting up your own static routing tables are not recommended because the installation of future revisions of the UNICOS system can overwrite the `/etc/staticrts` and/or the `/etc/tcpstart` scripts with updated versions. If you have chosen either of these methods, you must take the necessary steps to preserve the routing information that is contained therein as a local configuration modification.

2.2.9.8.2 Using Your Own Procedures to Set up Routing

If you are not using the `/etc/tcpstart` script supplied with the UNICOS system, your startup procedure must arrange to initialize the kernel routing tables with the appropriate routing configuration for your network topology.

You may elect to use the full dynamic routing capabilities of `gated`(8), in which case you should place the routing configuration for the `gated` daemon in the `/etc/gated.conf` file (see Section 2.2.8.1, page 61), and start `gated` as part of the TCP/IP daemon process (see Section 2.2.9.9, page 116, for information about starting daemons).

If you do not need the dynamic routing capabilities of `gated`(8), you can define static routing information in the `/etc/gated.conf` file and use the `/etc/staticrts` script to initialize the kernel routing table with this information. Alternatively, you can place `route`(8) commands in your start-up script to add routes directly to the kernel routing table.

2.2.9.9 Setting up Daemons

The `/etc/config/daemons` file contains a list of processes, known as *daemons*, necessary for system operation. To build the file, use the `Configure systems`

-> `System daemons configuration` -> `System daemons table` menu. In the sample `/etc/config/daemons` file, the columns are as follows:

Column	Description
`group`	Indicates the product to which the process belongs
`tag`	Indicates the name by which you want the process to be known
`start`	Indicates whether the process is to be started at system startup
`kill`	Contains the name of the file that contains the process ID of the currently executing daemon, or a hyphen (`-`) that indicates "do not kill," or an asterisk (`*`) that indicates "perform `ps	grep` *pathname* to find the process ID"
`pathname`	Indicates the command to be executed to start the daemon
`arguments`	Indicates the arguments to be used with the command in the `pathname` column

```
# /etc/config/daemons
#
# Configuration file for TCP daemons (and other commands) started by
# /etc/tcpstart.
#
# File format is:
#
# group tag        start      kill          pathname    arguments
#
TCP     gated       YES   /etc/gated.pid   /etc/gated   /usr/tmp/gated.log
TCP     named       YES   /etc/named.pid   /etc/named
TCP     inetd       YES   -                /etc/inetd   /etc/inetd.conf
TCP     talkd       NO    *                /etc/talkd
TCP     sendmail    YES   *                /usr/lib/sendmail  -bd -q30
```

`/etc/tcpstart` calls the utility script `/etc/sdaemon` to start the daemons. The `sdaemon` script consults the `/etc/config/daemons` file and starts the daemons that are listed there.

If you are not using the default `tcpstart` script, execute the daemons in your `tcpstart` script.

If you are using the default `tcpstart` script, check the `/etc/config/daemons` file to ensure that all appropriate daemons are listed. Comment out any daemons that should not be run at your site; add any daemons that your site needs.

The `/etc/inetd.conf` file contains a list of the daemons that `inetd`(8) can start. Check this file to ensure that all appropriate daemons are listed. Comment out any daemons that should not be run at your site.

The daemon processes that can be set up are listed in Section 2.2.8, page 61.

2.2.9.10 Calling the Final Local Script

The last process that `/etc/tcpstart` performs is to call the local script `/etc/tcpstart.pst`, if it exists and is executable. You should place in this script the following network-related processes:

- Processes that are specific to your local system
- Processes that must be performed after the networking software is started
- Processes that cannot be accomplished by any of the other mechanisms that are provided

2.2.10 Using the `telnet` Linemode Feature

The `telnet linemode` feature improves efficiency by moving line-oriented character processing from the Cray system to the front-end system. Usually, this process is transparent to the user. However, because of the distribution of character processing between the two systems, the user might see some minor problems in the following areas:

- Tab settings
- Special character processing
- Command completion and editing shells (`ksh` and `tcsh`)
- Simulated terminal input

2.2.10.1 Tab Settings

Users might notice some erratic screen activity when tabs are sent to their terminals. This activity is related to a difference in tab processing between the front-end system and the Cray system. Some screen-oriented applications, such as `vi`, appear to overwrite some characters randomly on the screen as the cursor moves through the text. Also, some commands might produce columns separated by tabs and one line (usually the first line of output) that is not aligned with the other lines. The `vi` editor can be affected in this way if line numbers are displayed.

You can address these problems by using the `stty`(1) command to check the default tab processing characteristics for the front-end system and Cray system `tty` drivers. The processing type should be set to either hard or soft tab settings; use hard tabs for best results.

The following example shows output from a common UNIX front-end system and a Cray system with the proper settings:

```
cunix> stty
new tty, speed 9600 baud; tabs crt pass8        <- "tabs" = hard tabs
pendin decctlq
start <undef>, stop <undef>, dsusp <undef>, lnext <undef>

Cray> stty
speed 9600 baud; -parity hupcl
eol <undef>;
-inpck icrnl onlcr      <- tab info appears here if not hard tabs
echo echoe echok
```

You can set hard tabs by using `stty tabs` command. If you cannot set both ends to use hard tabs, setting soft tabs (also known as *tab expansion*) on both ends is preferred. When using soft tabs, columnar output is sometimes imperfect. However, the worst formatting occurs when hard tabs are set on the Cray system and soft tabs are set on the front-end system; this situation should be avoided.

2.2.10.2 Special Character Processing

One aspect of supporting the `telnet linemode` feature involves special characters that are recognized by terminal drivers. When the front-end `telnet` program reads in a special character (such as an interrupt character), the front-end system must forward this character to the remote host for immediate processing. Unfortunately, many `telnet` implementations support an older `telnet linemode` feature, rather than the `telnet linemode` standard (RFC 1116). Although the older `telnet linemode` feature usually works well enough not to be noticeable to users, including the job control feature can cause the suspend character `CONTROL-Z` not to be forwarded to the Cray system.

You can solve this problem either by switching to character mode, or (the preferred solution) obtaining the latest copy of the `telnet` client.

You can set up character mode by giving a command to the `telnet` program to switch to character mode. Many `telnet` implementations accept the `mode character` command at their command prompt. (Check your front-end system's documentation for details.) Another method is to place the `stty -extproc` command in your login script (`.profile` for `/bin/sh` users and `.login` for `/bin/csh` users) on the Cray system. However, using character mode is not the preferred solution because it defeats the efficiency gained by making the `telnet linemode` feature available.

2.2.10.3 Command Completion/editing Shells (`ksh/tcsh`)

Using the `ksh` or `tcsh` shell removes the effect of the `telnet linemode` feature. Typically, these shells turn off echo and line editing and perform their own echoing and editing of the command line. Users who execute by using these shells cannot realize full advantage of the performance benefits of the `linemode telnet` feature.

2.2.10.4 Simulated Terminal Input

BSD UNIX has an `ioctl` procedure to simulate terminal input. This procedure is used by the Berkeley mail feature when you use `~h` to edit the mail header. It uses `TIOCSTI` to echo what you have typed so that you can edit it. Currently, the `telnet linemode` feature has no way of handling this facility.

2.2.11 Assisting Users in Setting up Environments

This section describes the following facilities for users:

- `$HOME/.netrc` file
- `$HOME/.rhosts`
- `bftp` facility

2.2.11.1 The `$HOME/.netrc` File

The `$HOME/.netrc` file is created by users and consists of the `ftp`(1B) and `rexec`(3) user authentication table. This file, located in a user's home directory, lists the host name, user name, and password for the user account on the remote host. When a user on the local host requests an `ftp` or `rexec` connection, the user's `$HOME/.netrc` file is searched and the user's login ID and password are automatically sent to the remote host. Consequently, the user is automatically logged in and is not prompted to provide either the login ID or the password.

Only official host names (not aliases) can be used in this file. If `$HOME/.netrc` does not exist, or exists but contains no entry for the host being accessed, the user is prompted for a login name and password.

Because this file contains password information, `ftp` and `rexec` require that it be accessible only to the owner; if it is not, an error message appears and `ftp` aborts. The *TCP/IP Network User's Guide*, contains additional information on `$HOME/.netrc` files.

The format of this file is as follows:

`machine` *hostname* `login` *userid* `password` *password*

hostname	Remote host's name as given in the `/etc/hosts` file. Aliases are not allowed.
userid	The user's login ID on the remote system.
password	The user's password on the remote system.

2.2.11.2 The `$HOME/.rhosts` File

The `$HOME/.rhosts` file, located in a user's home directory, is a private version of `/etc/hosts.equiv` and is used by the same servers as `/etc/hosts.equiv`. This file allows local accounts on the TCP/IP host to be used by specific remote users from specific remote hosts. This is especially helpful if users' login names on their remote hosts differ from their login names on the local host, or if you choose not to include all hosts in the `/etc/hosts.equiv` file. You do not maintain this file; however, you can supply a new user with a default `.rhosts` file.

If you place a remote user's login name in the second field of `/etc/hosts.equiv`, the user's `$HOME/.rhosts` file is not checked when the `-l` option of the `rlogin` command is specified; instead, the remote user is given automatic access to the account of the flagged user. See Section 2.2.5.3, page 37, for the details of setting up the `/etc/hosts.equiv` file.

The format of the `.rhosts` file is as follows:

official_host_name login_name `#optional comment`

The following provides an example of a `.rhosts` file for a user named Bonnie. Notice that Bonnie's login name on host `twg` is different from her login name on the other hosts.

```
# .rhosts file for Bonnie
#
cray1   bonnie
cray2   bonnie
twg     bon
```

If the file is writable by anyone other than the owner, the autologin information is ignored. For more information about the rules for setting up a `.rhosts` file, see the *TCP/IP Network User's Guide*.

Note: Additional restrictions are imposed when you are running with UNICOS security. See Section 2.6, page 211, or the *TCP/IP Network User's Guide*, for more information.

⚠ **Caution:** If the super user on the Cray system includes the remote host's name and the login name `root` in a `.rhosts` file that is created in the root directory of the local host, the root user from the remote host gains automatic authentication to the root account of the Cray system.

2.2.11.3 The `bftp` Facility

The `bftp` facility is an interactive user interface for batch network file transfers. The `bftp` program collects and queues information that is required for each file transfer, and a file transfer service (FTS) manages the file transfer for each request as the host becomes available on the network. After the user provides the appropriate information, the request is queued and the user is free to continue working without waiting for the completion of the file transfer.

The `bftp`(1B) command is most commonly invoked by a user already logged on to the Cray system, as follows:

```
prompt> bftp sncray
```

It is also possible to access `bftp` from a remote site by using the `-B` option of the `telnetd`(8) command. This is accomplished by running the `telnet` server `telnetd`(8), on an alternate port (one other than the normal `telnet` port `23`). Using `telnet` for remote access to that port number allows the user to log in and access the `bftp` command, rather than a shell.

For example, to support remote `bftp` access on port `50`, the `telnetd` entry in the `inetd.conf` file would be as follows:

```
50  stream  tcp  nowait  root  /etc/telnetd  telnetd -B 50
```

The `telnet` command to access `bftp` would be as follows:

```
prompt> telnet sncray 50
```

2.3 Network Tuning

Tuning is the process of adjusting the configuration of a system to optimize its efficiency. Tuning the UNICOS network software on your Cray system lets you achieve the maximum performance possible from your computer network and the Cray system.

The network software configuration that is included with the UNICOS system works effectively. It is designed to adapt, as much as possible, to the system's capabilities and to use those capabilities efficiently. However, the system configuration imposes limits. You must set the configuration properly to allow the software to make the best possible use of the hardware and to operate the network at its peak efficiency.

This section explains how to tune UNICOS network software parameters to achieve maximum performance. Although the default parameters included in the software might work well and require no tuning, by using this guide you can determine the changes, if any, you should make to optimize your system's network performance. To help you achieve performance gains, this section explains how the adjustable components of the system work and how you can affect their operation by making changes in the configuration information. These adjustable components include the following categories:

- Data transmission unit size
- Buffering and memory requirements
- Network routing

Each of the preceding has an enormous effect on network performance.

2.3.1 Data Transmission Units

Network software transfers data from one peer to another in data transmission units called *datagrams*. Each datagram consists of user data and a header that contains information used by the network software. IP carries datagrams between hosts on the network. TCP datagrams (known as *segments*) and UDP datagrams are carried in IP datagrams.

The network carries datagrams in *packets*, which consist of a header used by the hardware and the data to be carried in the packet. This data usually consists of higher-level datagrams, such as IP datagrams.

The maximum size of packet data that the network hardware can transmit at one time is known as the *maximum transmission unit*, or simply *mtu*. For TCP/IP, the packet size includes the segment size plus the 40 bytes of TCP and IP headers. For example, because the Ethernet can transmit 1500 bytes at a time, the Ethernet's mtu is 1500. The maximum TCP/IP segment an Ethernet can carry is 1460 bytes (1500 bytes minus 40 bytes for the TCP/IP headers).

You can control the actual size of the datagram, which lets you optimize network performance. Because the headers are of a fixed size for a given connection, there is a fixed amount of processing for each datagram, regardless of the amount of user data it contains. Network performance is generally improved by sending the largest possible datagram over a given network connection.

2.3.1.1 Interface Mtu—Model E and Model V Based Systems Only

Use the `hycf.` *name* configuration files and the `hyroute`(8) command to set the mtus of various directly connected hosts. An `hycf.` *name* file can be created for each network interface that is available to the Cray system. The `hycf` file controls the interface write mtu; the `ifconfig` command controls the interface read mtu. The `hycf` file is later used as input to the `hyroute` command, which is invoked by the `/etc/tcpstart` file at boot time. Each entry or host contains an mtu value. You can set the mtu value based on the mtu of the other hosts in the network. You can obtain this value by issuing a `netstat -i` command (or its equivalent) on the specific front-end system.

2.3.1.2 Using the Interface Mtu—GigaRing Based Systems Only

Use the `/etc/config/interfaces` *mtu* parameter to modify an interface's mtu and the `ifconfig`(8) command to set the mtus of various directly connected hosts. Use the route command to set the write mtu for a directly connected host, as follows:

```
route add -interface host -link hardware.address  admmtu mtu
```

The `ifconfig` command controls the interface mtu, and is invoked by the `/etc/tcpstart` file at boot time.

2.3.1.3 Datagram Size Limitations

The two characteristics of a network that limit datagram size are the mtu of the network medium and the size of the largest datagram that a host on the network can process. It is possible for a host to be connected to a network on which the mtu is too large for the host to process. Also, the IP protocol limits the size of an IP datagram. The largest allowable IP datagram is 65,535 bytes.

Because efficient use of the network relies on using the largest possible datagram size, both TCP and IP code contain algorithms to determine the datagram size to use for a given transfer.

2.3.1.4 IP Datagram Size Selection

IP protocol uses a very simple method to determine the size of IP datagrams to be transmitted; it attempts to send the largest datagram that the outgoing network interface can carry. If the datagram size is smaller than the interface mtu, the datagram can be transmitted in one piece. If the datagram size is larger than the mtu, IP protocol breaks the datagram into pieces known as *fragments* that are smaller than the mtu. When the destination host receives the fragments, they are reassembled into one IP datagram, and processing continues as though the original datagram were not fragmented. The protocol specifies that some datagrams must not be fragmented. When IP receives such a datagram, and the interface mtu is too small, IP discards the datagram and returns an error to the sender.

The mtu for each network interface should be set as large as possible when the interface is configured. Refer to Section 2.3.2.2, page 140.

2.3.1.5 Path Mtu Discovery

The UNICOS implementation of path mtu discovery conforms to RFC 1191. Path mtu discovery allows the network protocols to perform automatic mtu sizing. Path mtu discovery is activated by default; however, it can be disabled with the `netvar` command.

Path mtu discovery extends the IP mtu selection by using the IP `don't fragment` flag. Because of the nature of the mtu discovery mechanism, mtu discovery works only with TCP/IP; however, mtu information that is discovered by TCP/IP can be used by UDP and IP protocols. When sending data, TCP/IP uses the path mtu that is stored in the routing table for a particular destination as the segment size. The default size for the path mtu is the interface mtu that will be sent to the next hop gateway. The `don't fragment` bit is set in the IP header, and if an intervening gateway returns an error message, which indicates that the datagram must be fragmented, IP lowers the mtu for that destination until an mtu that is small enough for the gateway is found. Gateways that support path mtu discovery return the correct size for the mtu; for gateways without path mtu discovery, IP uses a table of common network mtu sizes.

By using path mtu discovery, a desirable mtu can be found for TCP/IP connections so that IP fragmentation can be avoided, regardless of the mtu of the network interface. Although path mtu discovery ensures that TCP/IP does not send segments that need to be fragmented, it does not ensure that the segment size is optimal for the connection. See Section 2.3.2.2, page 140, for an analysis of other considerations for the mtu configuration.

2.3.1.6 TCP Segment Size Selection

TCP datagrams are known as *segments*. TCP protocol attempts to select a segment size that is based on information about the interface, the route, and the host that is receiving the data.

If no useful information is available to determine an optimal segment size, TCP/IP uses a segment size that is guaranteed to be acceptable to the receiving host. The value of `TCP_MSS` is commonly used. This value is derived from the IP protocol that is specified in RFC 791, which specifies that every host that supports IP must be able to process an IP datagram of at least 576 bytes. This minimum size includes both the IP and the TCP headers; therefore, the actual minimum number of bytes of data is 536 (`TCP_MSS` = 576 - 20 - 20 = 536). Some systems decrease this number to a smaller value, such as 512, which leaves enough room for unconventional TCP and IP headers that might each be larger than 20 bytes.

The general algorithm used by the TCP kernel software to select its desired segment size is as follows:

1. Determine the route to be used. If no route can be determined, use `TCP_MSS`.

   The basis for selecting a segment size is the route. The route specifies which interface is used and which host on the network receives the packet. If no route can be determined, no useful information is available; consequently, a segment size of `TCP_MSS` is the only value that is guaranteed to be acceptable to the receiving host.

2. Determine the mtu of the route if a route can be determined.

   On a Cray system, the mtu is obtained by querying the driver for the mtu of the network that receives the transmitted packets. If path mtu discovery is enabled, TCP/IP offers the write interface mtu as the segment size. Then path mtu discovery is used to find the best mtu for the path if a smaller mtu is necessary due to fragmentation requirements at intervening gateways.

3. Compare the mtu specified for this route with the mtu of the directly connected network; when different, use the smaller value.

   (If an mtu is specified for the route, it is specified by the `route`(8) command.)

4. Determine if an mtu is specified for the route; if not, but the receiving host is on a directly connected network, use its mtu.

5. Otherwise, compare the mtu of the directly connected host acting as a gateway and `TCP_MSS`, and use the smaller of the two. The mtu of the directly connected host should not be smaller than `TCP_MSS`, but if it is, use the host's mtu.

   If the datagram must pass through a gateway, it is possible that the datagram will be sent to networks and hosts of unknown capabilities. Therefore, `TCP_MSS` is the only value you can use to guarantee that the segment size will be correct. Section 2.3.1.6.1, page 127 provides more information to determine whether a host is directly connected.

2.3.1.6.1 Subnetting and Direct Connections

Your Cray system might not be directly connected to the host, but it might still use the write interface mtu size for its segment size rather than `TCP_MSS`. Many networks are set up so that all of the local networks are subnets of one internetwork. A subnet uses part of the host portion of an Internet address as an extension of the network address.

Cray TCP/IP code is compiled so that two computers on different subnets of the same network are considered directly connected for the purpose of TCP segment size selection (more information on subnets can be found in RFC 950 and in Section 2.1.2.2, page 10). This means that if the two hosts have the same network number (allowing for subnetting), the TCP segment size selection algorithm uses the write interface mtu rather than the default size of `TCP_MSS`. If the two hosts have different network numbers, the connection uses the segment size `TCP_MSS`. You can continue to override the use of the `TCP_MSS` value by using the `route`(8) command to create a route with a specified mtu or by using dynamic mtu discovery.

Subnets that are to be treated as directly connected for the purpose of segment size selection are known as *local* subnets. However, under certain circumstances such as when various subnets are connected using network media of varying mtu sizes, you might want the segment size to default to `TCP_MSS`; that is, you might not want the subnets to be considered local. If a subnet is not local, the mtu of the directly connected interface is not considered to be the same as the subnet, and the segment size on the route defaults to `TCP_MSS`.

Use the `netvar`(8) command to change the `SUBNETSARELOCAL` kernel variable. You can change this variable temporarily at any time by using the `netvar` command, or you can permanently change this variable by changing your `netvar.conf` configuration file. See Section 2.2.9.2, page 107, for more information.

2.3.1.6.2 Segment Size Acceptance and the TCP MSS Option

When a TCP/IP connection is starting up, TCP/IP hosts (including Cray systems) send out the TCP maximum segment size (`MSS`) option (different from `TCP_MSS`), using the value derived from the TCP segment size selection algorithm. The `MSS` option indicates the largest segment that the host is willing to receive. It may reflect the limits of the interfaces and networks that the host is connected to, or it may reflect only the buffer space available in the host. `MSS` does not have to be the same for both directions of a TCP/IP connection.

The TCP/IP segment size exchange can negate the effects of path mtu discovery. If the peer TCP/IP determines that it should offer a segment size that is less than its own interface mtu, that is the segment size that the Cray system uses for the connection. To ignore the peer's suggested segment size violates the TCP/IP protocol.

Many systems do not implement path mtu discovery and suggest a small segment size (usually `TCP_MSS` = 512 or 536 bytes) when connecting with nonlocal networks. In this case, path mtu discovery has no effect. You can determine whether this is occurring by using `netstat` with the `-vv` option. The `Maxseg` column indicates the segment size of each TCP/IP connection; the `Peerseg` column indicates the size of the segment that the peer TCP/IP offers when the connection is established.

2.3.2 Buffering and Memory Requirements

Various parts of the network software operate by using memory to buffer data. The network software should have enough memory so that individual components can operate efficiently; however, it should have no more memory than necessary, because extra memory that the network does not use is also unavailable for other purposes. The UNICOS system is compiled with limits on the amount of memory that the networking software can use and limits on the size of buffers that can be used when transmitting and receiving user data.

If you are using the UNICOS ICMS, you can use the `Configure System -> Kernel configuration -> Network parameters` menu to change these values when you configure your system.

The TCP protocol ensures reliable transmission by storing user data before transmitting it, so that the data can be retransmitted, if necessary. TCP does not free stored data until the receiving host acknowledges receipt of the data. A TCP connection can be viewed as a pipe with a specific capacity. The capacity of the pipe varies, depending on the average time it takes for a host to acknowledge the receipt of data. If the pipe is not kept full, data is not being transferred at the highest possible transfer rate. If TCP software cannot buffer enough user data to keep the pipe filled, the network is not operating at peak efficiency.

Alternatively, if all active TCP connections are buffering a large amount of data, there might not be enough memory allocated to network usage. If this happens, some connections are forced to wait for memory to become available before transmitting data. If there is not enough memory for all the users of the network (using TCP, UDP, IP, and NFS applications), the network software cannot operate at peak efficiency. Therefore, buffering and total memory requirements must be set so that network software can operate at maximum efficiency.

Two types of memory parameters can be configured in the TCP/IP network software. (These configurations also affect the NFS kernel software because it shares the TCP/IP memory.) These parameters specify the total amount of memory available for use by the TCP/IP and NFS software, and the maximum amount of data that can be buffered at individual sockets.

2.3.2.1 Buffered Memory (Mbufs)

The TCP/IP and NFS network software uses memory from a special memory pool. The size of this pool is built into the kernel, and memory is allocated when the system is initialized (at boot time). The memory in this pool is grouped into pieces called *mbufs*. All data that is buffered by the network software and all dynamic data structures are stored in mbufs. The size of the mbuf pool is the limit placed on the total amount of system memory that can be used by TCP/IP and NFS.

2.3.2.1.1 the Mbuf Pool

When the Cray system boots, one part of the network initialization is to allocate memory for the mbuf pool. The quantity of memory that is allocated depends on a constant that is compiled into the kernel from the configuration file `/usr/src/uts/cf/config.h`. This constant, `TCP_NMBSPACE`, is the number of 1-Kbyte mbufs to be allocated for use by the TCP/IP and NFS network software.

If you are using the UNICOS ICMS, you can change this value by using the `Configure System -> Kernel configuration -> Network parameters` menu. This is the absolute maximum amount of memory that is available to the network software for data buffering and storage of dynamic data structures. If this number is set too low, network performance suffers. However, after this memory is allocated from the system, it is not available to user processes. Therefore, if you allocate excess memory to the network software, user processes (particularly large processes) are deprived of some memory. The following sections describe the criteria to use to allocate memory.

The network software uses mbufs for two purposes. The first is that user data is copied into mbufs pending transmission, and data that is read from the network is read into mbufs for storage until it is read by a user process. This use is related to socket buffering, which is discussed in Section 2.3.2.1.4, page 135. The second use is to keep track of dynamic data structures. Sockets, routing table entries, NFS user ID maps, and other structures are kept in mbufs.

2.3.2.1.2 Effects of Insufficient Mbuf Allocations

Insufficient mbuf allocations can affect network performance. If mbufs are not available when user programs are writing data to the network, the programs must wait until mbufs are available (unless they are using nonblocking I/O, in which case a write error occurs). If kernel-level software that is processing input data from the network requires mbufs and none are available, data is lost and requires retransmission. If user processes are trying to write data to the network and must wait for mbufs to be available, this wait can become the limiting factor in the performance of the network connection. If this is happening to several users, they will perceive the network (and possibly the Cray system itself) as being slow, or even down, when it actually is not.

When the network kernel software that processes incoming data from the network needs memory, it cannot wait for mbufs to become available. If the hardware drivers do not have memory available for network packets, those packets are lost. If the protocol-processing code needs more buffer space and cannot get it, the only alternative is to drop the data that is being processed. This loss of data has a negative impact on performance because data that is lost must be retransmitted. Also, there is a delay before the sender decides that the data is lost and requires retransmission.

Before refusing an mbuf request due to lack of available mbufs, the mbuf code in the kernel flushes the mbuf pool. During the flushing process, the mbuf code retrieves all possible mbufs from the areas that are set aside to hold specific sizes of mbufs (known as *private queues*), and it tries again to satisfy the request. Then, if not enough mbufs are available, the protocols are asked to drain any mbufs that are not absolutely necessary. For example, the IP protocol frees mbufs that contain fragmented IP datagrams that are not yet fully reassembled. This data needs to be retransmitted. This collection and compaction process (flushing and draining) takes time and can result in loss of some data. If there is still not enough memory to satisfy the request, the request is denied.

A shortage of mbuf space can have a very serious impact on the performance of the network software; the amount of space should be monitored to ensure that enough memory is available. If the system is flushing or draining the mbuf pool, performance is diminished. If requests are being denied, too little memory is allocated to mbufs. If flushes and drains are occurring, but no requests are being denied, there is enough memory for the network to operate, but not enough for the network to operate at maximum performance. Fortunately, you can easily determine the optimal number of mbufs to allocate; this procedure is described in the following section.

2.3.2.1.3 Mbuf Allocation and Monitoring

Using `netstat -m` is an easy method to determine the number of mbufs your system requires. You can also use the `netstat` command to determine all of the facts about how mbufs are being used on the Cray system. This command tells you how many mbufs are in use, and how many are being used for each method of mbuf allocation. It also gives valuable statistics on request denials and mbuf pool flushing. Following is an example of output from a `netstat -m` command:

```
% netstat -m
941/1800 mbufs in use (1152 max), allocated as 625 mbuf clusters:
      155 mbufs allocated to data (308 maximum)
      0 mbufs allocated to packet headers (80 maximum)
      101 mbufs allocated to socket structures (117 maximum)
      101 mbufs allocated to protocol control blocks (117 maximum)
      22 mbufs allocated to routing table entries (22 maximum)
      0 mbufs allocated to fragment reassembly queue headers (1 maximum)
      1 mbufs allocated to socket names and addresses (25 maximum)
      0 mbufs allocated to socket options (1 maximum)
      4 mbufs allocated to interface addresses (4 maximum)
      0 mbufs allocated to NFS notify-when-free data areas (1 maximum)
      180 mbufs allocated to NFS static data (180 maximum)
      368 mbufs allocated to NFS dynamic data (368 maximum)
      9 mbufs allocated to system buffer headers (29 maximum)
```

```
0 mbuf queue flushes
0 protocol mbuf queue drains
0 requests for memory denied
%
```

The first line of output from `netstat -m` has four pieces of information. The first two are the number of mbufs in use at that moment and the total number of mbufs available, expressed as `941/1800 mbufs in use` in the preceding example. The next bit of information, `1152 max`, indicates the largest number of mbufs that were ever in use at any one time. This is the maximum number of mbufs in use at some point in time since the system was last booted. Finally, this line tells you how many clusters are in use at the current time. A *cluster* is a group of mbufs being used as one large mbuf.

The indented lines from the `netstat -m` display tell how many mbufs are in use for each of a variety of purposes. You can also see the maximum allocations that have occurred for each of these purposes.

In the last three lines of the display, you can see how many times the mbuf pool was flushed, and how many times the protocols were asked to drain mbufs that were not absolutely necessary. You can also see how many times an mbuf request was denied due to lack of available mbufs. The number of flushes is always greater than or equal to the number of drains, and the number of drains is always greater than or equal to the number of denials. The relationship between flushes, drains, and denials can be stated as follows:

flushes >= drains >= denials

The value of flushes minus drains tells you how many times flushing was successful at getting mbufs to satisfy a request. The value of drains minus denials tells you how many times draining was successful at getting mbufs to satisfy a request.

Five of the statistics that `netstat -m` provides are very valuable for determining the number of mbufs your system requires. These are the number of mbufs available (the second number in `941/1800`) and the maximum mbuf usage (`1152 max`) from the first output line, and the number of flushes, drains, and denials. The usefulness of these statistics is directly proportional to the length of time since the system was last rebooted. The longer the time, the more useful the statistics.

These five numbers indicate the location of your system in the performance chart shown in Figure 8.

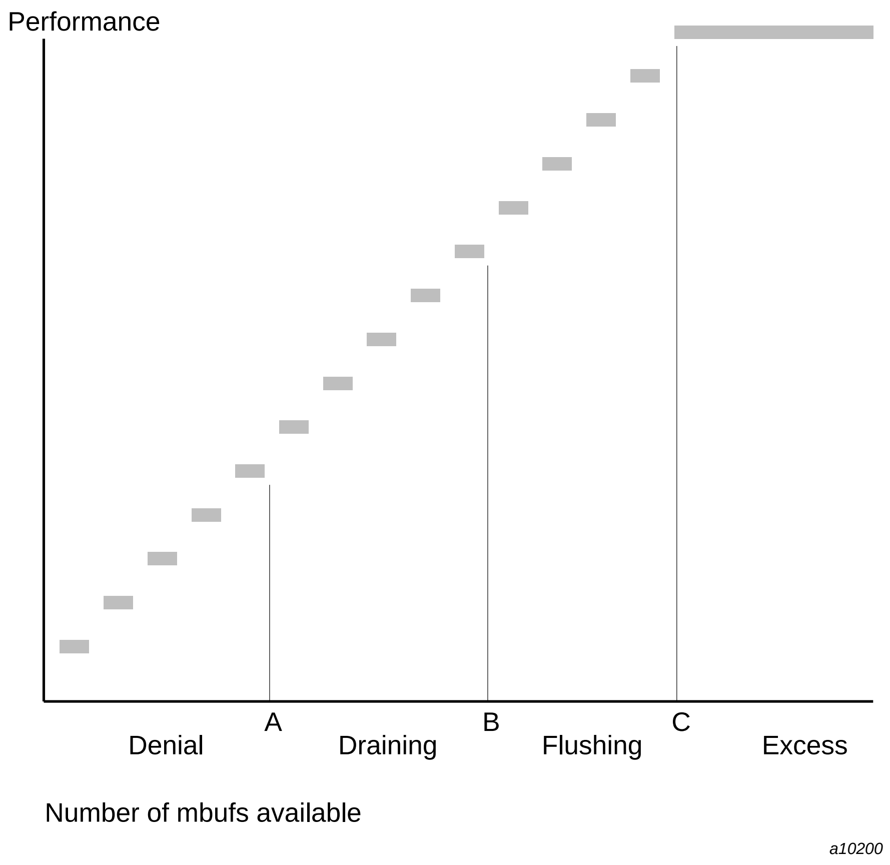


Figure 8. Performance chart

You will notice from the graph that adding mbufs improves network performance up to point C, where adding mbufs has no further effect. At this level, there are no flushes, drains, or denials, and maximum mbuf usage is equal to the number of mbufs available. The system has exactly as many mbufs as it needs. Point A on the chart is the point at which the system crosses the boundary from denying requests to simply flushing and draining to satisfy requests. Point B is the point at which draining ceases and only flushes are occurring.

You can determine the location of your system in this chart by looking at the five statistics. If any requests were denied, your system is on the bottom quarter of the chart. If draining occurred, but no requests were denied, your system is on the second quarter of the chart. If flushes occurred, but no draining occurred, your system is on the third quarter of the chart. If maximum mbuf usage was less than the number of mbufs available, your system is on the top quarter of the chart.

Gains from adding mbufs are more significant as you go down the scale. In the bottom half of the graph, data is being lost, and requires retransmission. (Whenever a drain occurs, data is lost.) A drain that is followed by a denial means that both data already received and incoming data are lost, which is a very significant performance loss. Flushing is much less significant, because the mbuf code simply clears its queues of presized mbufs; then no data and only a small amount of time are lost.

Using this graph and `netstat -m`, you can determine how mbuf allocation is affecting network performance, and you can identify the optimal mbuf allocation for your system.

The optimal level for your system depends on how your system is used. The only way to determine this is by trial and error; however, you can speed up this process by configuring more mbufs than you think you will need. Use `netstat -m`, and watch the machine's statistics. Over time you will develop an understanding of the mbuf usage. To calculate, take the highest mbuf usage and round it up to the nearest 100 mbufs (an extra buffer of safety). Then reconfigure your kernel to use this number of mbufs. You can do this by changing the value of the `TCP_NMBSPACE` parameter in the `/usr/src/uts/cf/config.h` file, or, if you are using the UNICOS ICMS, by setting the appropriate value on the `Configure System -> Kernel configuration -> Network parameters` menu. See Section 2.3.2.1.3, page 131, and Section 2.2.2.2, page 28, for more information on selecting initial mbuf pool size and changing this parameter.

If you have a system with small memory, you might want to consider making a trade-off between network performance and memory consumption. Performance decreases greatly when drains and denials are occurring, but a worthwhile trade-off can put your system into the flushing range of mbuf allocation.

If HIPPI or GigaRing is used as a network interface, you must allocate more than the usual 1800 mbufs for best data rates. A range of 3500 to 10,000 mbufs might be required, depending on how heavily the HIPPI or GigaRing interface is used.

For all interfaces, the number of mbufs used also depends on the values of rbuf and wbuf.

2.3.2.1.4 Sockets and Socket Buffers

Sockets are end points of communication between two hosts; that is, they are the points from which data is sent and at which data reaches its destination. *Socket buffers* (also known as *sockbufs*) are clusters or chains of mbufs that hold the data being transmitted and the data to be received; they are always defined in pairs. All of the buffering at a socket is done in these two associated data structures. There is a sockbuf for data being transmitted (the *send sockbuf*), and another for data being received at the socket (the *receive sockbuf*). When data is to be buffered in a sockbuf, an mbuf containing the data is added to a list of mbufs maintained in the sockbuf.

Each sockbuf has an associated buffer limit. Defaults for these limits are built into the kernel, but these limits can be changed. The `/usr/src/uts/include/sys/tcp_config.h` file specifies the default limits for TCP and UDP sockets. The `SOCKETSEND` and `SOCKETRECV` parameters specify the default sockbuf size limits for TCP. The `UDPSENDSPACE` and `UDPRECVSPACE` parameters specify the default sockbuf size limits for UDP. The `SOCKBUF_MAX` parameter specifies the maximum size limit that socket applications can specify through the `SO_SNDBUF` and `SO_RCVBUF` socket options. The `MAX_SOCKETS` parameter specifies the maximum number of open sockets.

At run time, you can set these values by adding `/etc/netvar` with the appropriate options and values to your `tcpstart` script. Selection of values for these parameters and information about how to change them are discussed in succeeding paragraphs.

The size limits on sockbufs can affect performance in more than one way. Not only do the limits restrict the amount of data that can be buffered on a socket, but the protocols also use this information.

Unlike UDP, which simply sends datagrams without considering the buffering capacity at the other end, the TCP protocol has built-in features to optimize its efficiency. Also, the Cray TCP protocol software has other optimizing features, not specified by the protocol, that further improve performance.

The TCP protocol informs the sending host about the amount of buffering that is available at the socket. One way the TCP protocol attempts to improve performance involves keeping track of the amount of data buffering space that is available at the receiving host. TCP does not send data to a host that the TCP protocol determines does not have available buffer space. This limitation improves efficiency by avoiding a waste of network bandwidth on data that must be discarded by the receiver (due to lack of space) and then resent.

The TCP protocol is a reliable protocol, and it ensures that data is transmitted and received. TCP accomplishes this by saving a copy of all data transmitted until it receives an acknowledgment from the other end of the connection that data is received. There is a limit on the amount of data that TCP/IP buffers while awaiting acknowledgment. This limit is the size of the send socket buffer.

Another way that the TCP protocol software attempts to efficiently use network bandwidth is to send only full segments of data (instead of fragments) when reasonable. A segment is the largest amount of data that can reasonably be sent and received in one datagram. This helps to reduce the number of packets sent on the network.

Because of the optimizations built into the TCP software, the strategy for achieving optimal performance is to set configuration parameters so that they equal or exceed the upper bounds of maximum performance. The TCP software efficiently uses the available resources, but it does not tie up network resources unnecessarily.

One factor to consider when determining the best size for the socket buffers is that the limits that are set are not the amount of memory that is allocated to the buffers, but the maximum amount of memory that can be used to store data on the socket buffers. When a socket is created, its socket buffer size is initialized, but no memory is allocated for that socket buffer. The buffer size simply limits the number of mbufs that can be chained into the socket buffer’s mbuf list; therefore, a large socket buffer does not necessarily indicate that a large amount of memory is used for that buffer.

Choosing socket buffer size limits is not as straightforward as configuring your system for enough mbufs. This is because the sockbuf sizes are dependent on the buffering limits at the other end of the transmission. A reasonable choice is to set the default buffer sizes for TCP at 64 KB, but this is not optimal. Although most processes that use TCP read the incoming data quickly, some may not. Also, if a TCP sender has a very large send buffer, but the receiver has a small receive buffer, TCP fills the send buffer but gains little (if any) performance increase, thus wasting memory. Because of these possibilities, arbitrarily large TCP buffers might waste memory resources unnecessarily. (This is also true for UDP sockets.)

There are other reasons for not always setting the default buffer sizes for TCP at 64 KB. Because the buffer sizes are communicated between TCP peers, the peer knows the receive buffer size. Some TCP implementations do not calculate buffer sizes properly and do not work properly if the Cray buffer size is too large.

Choosing optimal buffer sizes is further complicated by the fact that the optimal buffer size is relative to the size of the received segments, the speed of the network, and the buffering available at the other end of the connection. For a connection between two Cray systems over a high-speed channel, the buffer sizes should always be 64 KB or larger (for example 128 or 256 KB), because large segments can be used, and the network is very fast. However, for a connection between a workstation and a Cray system, which goes through a front-end system, a smaller buffer size is acceptable.

Choosing optimal send and receive sockbuf size limits can be greatly simplified by considering the concept of double buffering. *Double buffering* is often used in I/O systems in which a process or hardware device is filling a buffer that another process or hardware device is reading. If there is only one buffer, one process must wait for the other to finish reading (or writing) before it can proceed. If there are two buffers, and the processes switch between them, one process can fill one buffer while the other process is reading the other buffer. Both processes can run at maximum speed at all times.

Consider the network shown in Figure 9. Suppose that the send and receive sockbuf limits at the workstation are 8 KB. One direct effect of this is that the workstation never has more than 8 KB of data in flight on the connection, where *in flight* means that the data is sent but not yet acknowledged. This is because the workstation can buffer only 8 KB of transmitted but unacknowledged data. Also, suppose that the receiving process running on the Cray system is reading the data as fast as it comes in. This can be faster than the data is being sent, when considering the relative speed of the Cray system as compared to most workstations that communicate over an Ethernet.

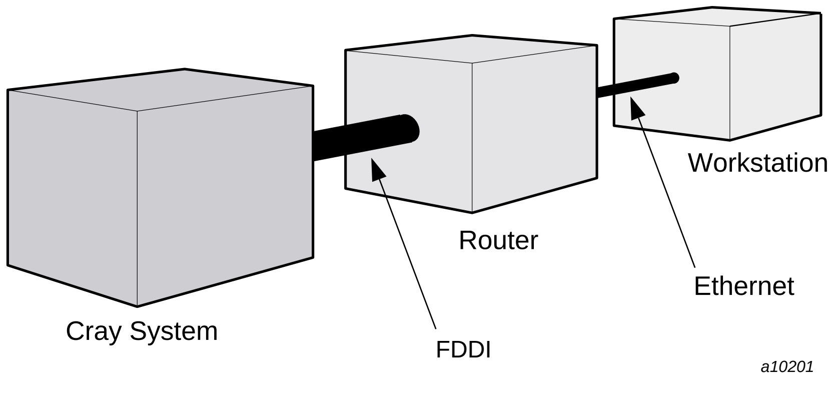


Figure 9. Double Buffering Network

When user data is read from the socket buffer, the data must be copied from the kernel receive buffer to the user's buffer. This takes time. If the kernel receive buffer is twice as large (double buffering) as the peer's send buffer, the peer can be filling one half of the receive buffer while the user is emptying the other half. In this situation, a receive buffer of 16 KB is sufficient to achieve maximum performance over the connection.

Similarly, a 16-Kbyte limit on the send sockbuf on the Cray system is also sufficient to run the connection at maximum performance. Because the workstation can never buffer more than 8 KB of data, TCP on the Cray system never tries to send more than 8 KB of data without an acknowledgment from the workstation. The extra-large send buffer ensures that data is available to send when the protocol is ready to send data; because there is no waiting for data to be copied from user memory to kernel memory, a 16-Kbyte send sockbuf limit is enough.

Consider another example that uses the same network. This example relates to TCP segment size. The Cray system sends a TCP `MSS` option for the mtu on the FDDI connection, which defaults to 4352 bytes. With 40 bytes subtracted for the TCP and IP headers, the request size is 4312 bytes. The workstation sends out a TCP `MSS` option of 1460 bytes, based on the Ethernet mtu of 1500 bytes, minus 40 bytes for TCP and IP headers. Some workstations suggest a convenient size of 1 Kbyte (or 1024 bytes) for segments on Ethernet connections (the convenience factor involves memory allocations). Suppose the workstation requests the convenient size of 1024, and the Cray system honors this request and sends segments of the requested size. When the workstation receives the Cray system's `MSS` of 4104 bytes, it determines that the mtu on its outbound interface is lower, and it uses its own segment size.

It is necessary that the buffer limits be at least twice the segment size (double buffering) being used for this connection. This allows some operations involved in the transfer of data to overlap or run in parallel. If the buffer limits were the same as the segment size, the connection would become a synchronous mode of operation. In synchronous mode, the sender sends a full buffer (buffer limit = segment size) of data, waits for acknowledgment before refilling the send buffer with new data to send, and then sends it when the receiver informs the sender that the receive buffer is clear and there is space for more data.

If the sockbuf size limits are twice the segment size, the sender can buffer and send a second segment while the receiver is processing the first segment. When the sender is ready to send another segment, the sender might have received notice that the receiver has room to buffer the segment; if so, the sender can send it immediately. Then, the sender is never waiting for buffer space to become available at the receiver. This asynchronous mode of data transfer is more efficient than the synchronous mode of data transfer; therefore, with a segment size of 1024 bytes, the buffer limits must be at least 2048 bytes.

If the segment size is the same size as the capacity of the Ethernet (1500 bytes), which is probable for workstations that do not round down the segment size to a convenient size, the buffer limits should be at least 3 KB. In practice, a buffer size of 3 or 4 times the segment size improves performance even more. Where possible, buffering limits should be 4 times the segment size. Therefore, for this example, the sockbuf limit should be 4 KB. If the buffer size that is calculated relative to the segment size differs from the buffer size that is calculated from the peer's buffer size, use the larger of the two sizes, if possible.

Finally, consider the possibility that the network connection is slow between the communicating peers. This has the effect of delaying the acknowledgment of data that comes from the receiver and also the notification that buffer space is available (data having been read by the user process). This delay could mean that the sender sent a full buffer of data and is waiting for an update, even though the receiver of the data may have sent the update. Ideally, it should take the sender the same amount of time to send a full buffer of data as it takes for the notification of available buffer space to get back to the sender. Assuming that you cannot do anything to increase the speed of the network, the solution is to make the receiver's buffer large enough to ensure that the sender does not have to wait for notification of available space. Of course, the sender's send buffer must be increased to take advantage of this. If you cannot change the buffer sizes on your systems that are not Cray systems, you might not be able to resolve the problems of a slow network. Fortunately, most local area networks are fast enough, so that this is not a problem.

In considering these factors, you should set the default TCP sockbuf size limits to the size required for your highest-performance connection. This is because all network applications use the default size that is compiled into the network kernel software, rather than each application using an optimal size for that particular connection. If you set the size limits smaller, the high-performance network connections do not work at their optimal performance level. Users expect large increases in performance over high-performance connections.

If your Cray system is in a network in which it does not have high-bandwidth connections, you can use lower default sizes in the kernel without decreasing performance.

You can change the sockbuf size limits on a per socket basis with the `setsockopt()` function. The `SO_SNDBUF` and `SO_RCVBUF` options are used to set the send sockbuf size limit and the receive sockbuf size limit, respectively. An example code fragment follows:

```
/* set sockbufs to 64k */
int sockbufsize = 65536;
/* set send sockbuf size limit */
```

```
setsockopt(sock, SOL_SOCKET, SO_SNDBUF, &sockbufsize, sizeof(sockbufsize));
/* set receive sockbuf size limit */
setsockopt(sock, SOL_SOCKET, SO_RCVBUF, &sockbufsize, sizeof(sockbufsize));
```

You can set the send and receive sockbuf size limits to varying values, if you want. However, this changes the buffer sizes for only a specific socket; no other sockets are affected.

2.3.2.2 A Network Example

This example illustrates how the Cray system can be configured to ensure that the optimal segment size is chosen. Remember that the segment size drops to the smaller of the two segment sizes that are suggested by the peer or calculated by the Cray system. TCP/IP on the Cray system should be set up to ensure that the segment size it calculates is as large as or larger than that suggested by the peer. A sample network configuration is shown in Figure 10.

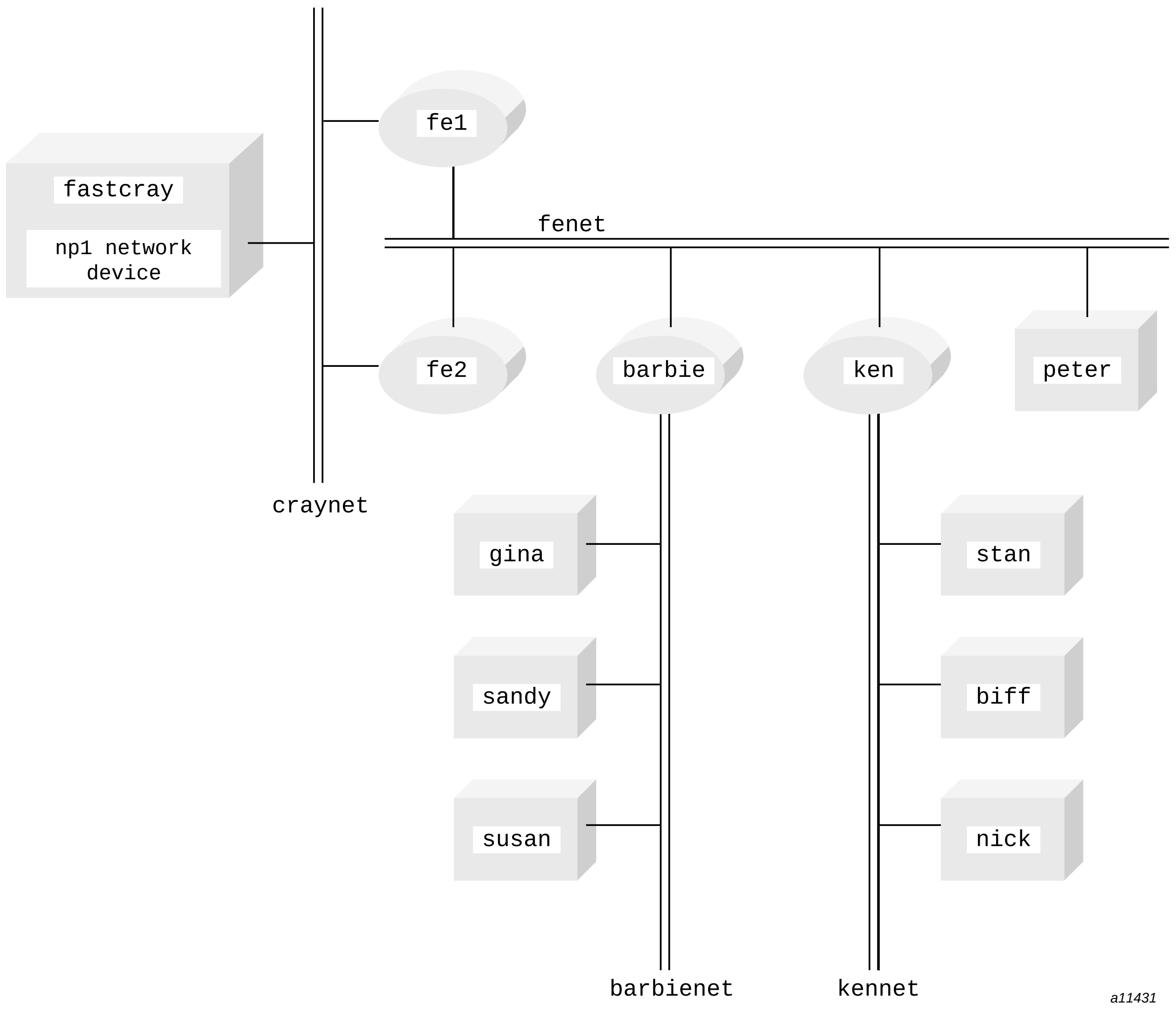


Figure 10. Sample network configuration

The system called `fastcray` is a Cray system, and `fe1` and `fe2` are front-end systems connected to it over HYPERchannel interfaces. `barbie` and `ken` are workstation servers, and `peter` is a stand-alone system. `fenet` is a 100-Mbps network medium with a 4478-byte mtu. `barbienet` uses the same medium and has the same characteristics as `fenet`, but `kennet` uses 10-Mbit/s medium with a 1500-byte mtu. These mtus are the size of the maximum datagram that can be transmitted, and network packet headers do not need to be considered.

Knowing the mtu of a network medium is not always enough. There are some situations in which an interface to a network cannot support the full mtu of the network. However, this is unusual and is not considered here. It is sufficient to note that if a network interface supports an mtu that is smaller than the mtu supported by the network medium, the interface mtu is the one that should be used.

2.3.2.2.1 Selecting the HYPERchannel Interface Mtus

The first step in configuring this network is selecting the mtu for the `fastcray` HYPERchannel interface. There are two mtus to consider: the mtu for outbound datagrams being transmitted on the interface, and the mtu for datagrams being read by the interface. The mtu for outbound datagrams is set to match the size of the datagrams that the peer interfaces can accept.

Calculating the inbound (read) mtu is important because driver read buffers are allocated in advance. The size of the buffers must be predetermined, and multiple buffers must be allocated in advance. If the buffers are larger than necessary, memory is wasted.

The default value for the read mtu on HYPERchannel interfaces is 16,432 bytes. This value can be changed on a per-interface basis with the `ifconfig`(8) command. The read mtu on an interface should be set to the size of the largest datagram that is expected through that interface.

Next, you must determine the interface mtus on `fe1` and `fe2`. Because the mtu on the HYPERchannel is not a fixed value, a reasonable value must be chosen; that is, the mtu must be set to a value that supports high transfer rates not only to `fe1` and `fe2`, but also to the other systems that use `fe1` and `fe2` as gateways to `fastcray`.

Because the HYPERchannel network does not suggest an mtu, you must consider other factors. For example, the mtu on other networks to which `fe1` and `fe2` are connected is important. You should also consider how the mtu affects the TCP segment size and the relationship of TCP segment size to socket buffering capacity. Obviously, the HYPERchannel mtus cannot be selected until the mtus of the other interfaces are established.

2.3.2.2.2 Selecting Mtus for the Other Networks

Consider the buffering and datagram-handling capacities of `fe1` and `fe2`. You cannot send datagrams that are larger than the largest datagrams that `fe1` and `fe2` can process. Usually, the limit on datagram size is the same as the socket buffering limits. For more information, consult the documentation for your specific systems.

Suppose that the socket buffering and datagrams limits are 8 KB for `fe1` and 12 KB for `fe2`. This places a hard limit on the size of the datagrams that can be transferred to these machines. The mtus must be selected so that the maximum size of datagrams sent to `fe1` is 8192 bytes or less, and the maximum size of datagrams sent to `fe2` is 12,288 bytes or less.

Next, consider the relationship of segment size to TCP socket buffering capacity. Section 2.3.2.1.4, page 135, explains why it is necessary for the socket buffer limits to be at least twice the segment size. Because of this, the mtu to `fe1` and `fe2` should be lowered to allow datagrams of only half the socket buffering capacity. This allows much higher performance on TCP connections between `fastcray` and the front-end systems. Therefore, selecting mtus for `fe1` and `fe2` of 4096 bytes and 6144 bytes, respectively, is reasonable.

2.3.2.2.3 Selecting Optimal Mtu Values

Using the values established in the previous section to determine the mtus for the front-end systems might not be sufficient for optimal performance. One consideration is that subnets can be used as local networks in the selection of TCP segment size.

In this example, if the `SUBNETSARELOCAL` kernel variable is not changed from its default value of `true`, and `craynet`, `fenet`, `barbienet`, and `kennet` are all subnets of the same network, the TCP segment size for hosts on those nets is selected as if the network mtu to those hosts were the same as the mtu to the gateway being used to communicate with the hosts. Therefore, if datagrams for `barbie`'s clients are routed through `fe2`, the TCP segment size is 6 KB, and `fe2` is forced to fragment the datagrams to send them out on `fenet`. Or, less desirable, if datagrams for `ken` 's clients are routed through `fe2`, `fe2` fragments the datagrams to transmit them on `fenet`, and `ken` must fragment the fragments so that they can be transmitted on `kennet`. This is not optimal efficiency.

If subnets are not being used, or are being used but you use either the `netvar`(8) command or the `sysctl`(8) command to change the `SUBNETSARELOCAL` variable to `false`, the other networks need not be considered, because the mtu on the interface is not used for the TCP segment size for nonlocal connections.

Assume that this example network is configured so that all of the networks are local subnets. You can control the segment size used for the local subnets either by changing the mtu on the interface so that sufficiently small packets are transmitted, or by using the `-admmtu` option of the `route`(8) command to affect TCP segment size selection.

2.3.2.2.4 Changing the Interface Mtu

First, select an mtu that allows for the effects of local subnets. The mtu in `fenet` (and the maximum datagram size) is 4478 bytes. To avoid fragmentation, datagrams traveling through `fe1` or `fe2` should not exceed 4478 bytes. Assume that the mtu on the `fenet` interfaces for `barbie`, `ken`, and `peter` is also 4478 bytes; therefore, fragmentation is not a factor. The socket buffering and largest datagrams that `barbie`, `ken` and `peter` accept are 8 KB, 8 KB, and 6 KB, respectively. These are large enough not to affect the mtus any further.

However, because you might want to set the mtu for `fe2` down to 4478 bytes (to avoid fragmentation), you should consider the effect on TCP performance of `barbie` ’s, `ken` ’s, and `peter` ’s socket buffering. Decreasing the mtu to half the size of the smallest buffer implies that half of `peter` ’s 6-Kbyte buffer is an appropriate size for the mtu. So, at this point, a 3-Kbyte mtu from `fastcray` to `fe1` and `fe2` appears correct.

Next, you should consider `barbie` ’s and `ken` ’s clients. Assume that `barbie` ’s clients have the same characteristics as `barbie`, and, because they are connected to `barbie` through the same high-performance network medium as `fenet`, `barbie` ’s clients have no further effect on mtu calculations. However, `ken` ’s clients are connected to `ken` through a slower network with a smaller mtu. `kennet` ’s mtu of 1500 bytes suggests that `fastcray` ’s mtu to `fe1` and `fe2` should be dropped to 1500 bytes. Small datagram handling capabilities and socket buffering at `ken` ’s clients might suggest an even smaller mtu.

Clearly, this is not an appropriate solution for selecting the mtu on the Cray system’s network interface.

Note: You cannot change the mtu size of the Cray J90 system Ethernet or FDDI interfaces.

2.3.2.2.5 Using the `route` Command

By using the `route`(8) command to specify an mtu for a route, you can specify the TCP segment size for that route. Then you can override the interface mtu and avoid choosing the small value of `TCP_MSS`. When adding routes to `fenet`, `barbienet`, `kennet`, and `peter`, you can set the mtu on the route so that the TCP segment size is calculated to an appropriate value. For this example, use the following `route` commands:

```
/etc/route add fenet fe2 -admmtu 4478
/etc/route add peter fe1 -admmtu 3072
/etc/route add barbienet fe2 -admmtu 4478
/etc/route add kennet fe1 -admmtu 1500
```

For Model E based systems, the appropriate lines in the `/etc/hycf.np0` file are as follows:

```
direct fe1 4233 ff00 0 4478
direct fe2 4543 ff00 0 6144
```

This solution allows TCP segments of optimal size to travel to all hosts in the network.

2.3.2.2.6 Inbound Mtu

Because the other mtu values are established, determining the inbound mtu for `fastcray` 's HYPERchannel interface is very easy. Set this value to the largest outbound mtu for the interface. This is the largest packet that a directly connected host is sending to `fastcray`. Read buffers of this size are sufficient to hold the largest packets received. In this example, this value is 6144 bytes.

2.3.2.2.7 More Optimizing Considerations

Neither of the previously described solutions is completely optimal for all situations. Consider the advantages and disadvantages of each method, and you will see that both methods are useful.

The advantage of using the interface mtu to control datagram size is that this control applies to all protocols. Consequently, all TCP, UDP, NFS, and IP datagrams can be small enough to avoid fragmentation as they travel through the network. Fragmentation, especially at a busy gateway, can have a severe performance impact. The mtu set with the `route` command affects only the TCP segment size and has no effect on UDP, NFS, or IP datagrams.

The disadvantage of changing the interface mtu is that this approach lowers performance on all connections by limiting the datagram size to the optimal size required for the connection with the lowest performance.

The advantage of using the `route`(8) command to control TCP segment size is that you can tune performance to each host's and network's capabilities. TCP uses appropriate segment sizes for each of the routes in the routing table.

The disadvantage of using the `route`(8) command to control TCP segment size is that you affect only TCP segment size, as previously described. NFS transfers can be affected by significant fragmentation delays over lower performance connections.

The advantage of having these competing goals is that you can often use the route mtu (an option for only Cray systems) to optimize TCP transfers and also to use the large hardware mtu. This is because systems offering high performance transfers are typically connected to the high performance (and large mtu) networks.

2.3.2.2.8 Policy for Decisions

After you have considered several factors that affect decisions about mtu size on the Cray system's network interface, you can determine a policy that helps you make appropriate decisions.

First, for this example, consider how `fe1` and `fe2` are to be used. Consider whether they are actual systems that are going to be used for actual computing, or simply gateways to serve the remaining network. If they are gateways, all they must do is move datagrams quickly between `craynet` and `fenet`. If they do other work, consider the quality of network performance between them and `fastcray`.

Also, consider how `barbie`, `ken`, and `peter` are used. Their functions determine priorities in the quality of their connections. An NFS server that contains files that will migrate to and from the Cray system must have a high-quality connection with interface mtu sizes that are large enough to support large datagrams. A server for a group of clients can also benefit from high-bandwidth connections, but in this case, file transfer speed to the Cray system might not be as important. A machine that is used only to support dial-up lines and not to store much data probably will not require a high-performance connection at all; perhaps the only access it will require to the Cray system is `telnet`(1B)) (to support logins to the Cray system).

In this hypothetical network, assume the following:

- `fe1` is used primarily as a gateway.
- `fe2` is a computer that supports many users, and therefore, it must not be overloaded with gateway work.
- `barbie` is a large fileserver, and it stores files that are used by many users from many locations.

- ken is a smaller server for its local clients.
- peter supports dial-up lines.

peter 's function as a telnet gateway to fastcray suggests that most transfers between peter and fastcray are small and they do not affect other decisions.

ken 's status is similar to peter's; this machine supports telnet connections from clients and an occasional NFS transfer. Because there are no large transfers from ken 's clients (diskless clients), fragmentation is not an issue. Of course, when a user uses ftp(1B) to transfer a file, that user must log in to ken, either directly or through the proxy command, before transferring the file to fastcray. It only slows down performance if the file must be pulled from ken to the client (using NFS), and then over TCP to fastcray.

barbie's status as a high-performance server suggests that it should have a high-performance connection for NFS transfers. This means that the datagrams that are transferred should be as large as possible.

Because fe1 is a gateway, the mtu to fe1 should be as large as fenet 's mtu; fe1 should not consider maximum datagram handling and socket buffering for datagrams that are larger than fenet's mtu. All routes from fastcray to the remainder of the network should pass through fe1.

fe2 should support a high-performance connection to fastcray for TCP users.

For Model E based systems, use the following guidelines:

- The write mtu to fe1 in the hycf configuration file should be 4478 bytes; the write mtu to fe2 in the hycf configuration file should be 6144 bytes. The appropriate lines to put in the /etc/hycf.np0 file are as follows:

  ```
  #direct MACH-NAME       TO                      MTU
  direct  fe1             4233    ff00    0       4478
  direct  fe2             4543    ff00    0       6144
  direct  fastcray        c205    ff00    0
  ```

  The inbound mtu for the np0 interface is set to 6144 bytes (the largest outbound mtu) when the interface is initialized with the ifconfig command, as follows:

  ```
  /etc/ifconfig np0 fastcray netmask 0xffffff00 mtu 6144
  ```

For GigaRing based systems, use the following guidelines:

- The write mtu to `fe1` in the `arp` route should be 4478 bytes; the write mtu to `fe2` in the `arp` configuration file should be 6144 bytes. The appropriate `route` commands are as follows:

  ```
  route add -interface fe1 -link hardware_address -admmtu 4478
  route add -interface fenet fe2 -link hardware_address -admmtu 6144
  ```

  The inbound mtu for the `ghippi0` interface is set to 6144 bytes (the largest outbound mtu) when the interface is initialized with the `ifconfig` command, as follows:

  ```
  /etc/ifconfig ghippi0 fastcray netmask 0xffffff00 mtu 6144
  ```

Appropriate routes to the world must be added. You should specify mtus for those routes on which TCP performance is improved by using a smaller segment size than the mtu allowed by the interface, as follows:

```
/etc/route add peter fe1 -admmtu 3072
/etc/route add fenet fe1
/etc/route add barbienet fe2
/etc/route add kennet fe1 -admmtu 1500
```

2.3.3 Network Routing

This section describes the `route`(8) command, but if you are using the UNICOS start-up procedures, you must use the `gated.conf` file, which offers the same functionality as the `route` command. See Section 2.2.9.8, page 114, for details concerning the use of the `gated.conf` file.

The `route`(8) command is useful for setting mtus and also gives you complete control of the routing table, allowing you to specify host, network, and default routes. The destination operand on the command line specifies whether you are adding a host, network, or default route. When a network address is given, a network route is added; otherwise, a host route is added. To determine this, the `route` command interprets a dot notation address (such as `174.200.84.132`), or it chooses between host or network based on which of `getnetbyname()` or `gethostbyname()` successfully resolves a symbolic name. `getnetbyname` is tried first; therefore, if you have a network and a host with the same name, you must use the `host` operand on the `route` command line to create a host route to that host. Similarly, the `net` operand indicates a network route. The `route` command also recognizes the destination name `default` as directing it to add a default route.

You can use your knowledge of how routing decisions are made (see Section 2.1.4.2, page 19) to affect these decisions. For example, a host-specific route always takes precedence over network and default routes. Also, after a route is found, no attempt is made to check for other possible routes. If a route that does not work appears in the table before a route that does work, the invalid route is used, and communication with that particular host or network is impossible.

You can configure the routing tables so that the routing algorithm chooses among multiple potential routes. However, not all network configurations are sufficiently complex to make this possible. Use the following techniques, when possible, to achieve specific goals:

- Force data destined for a particular host to take a different path from that of data going to other hosts on the same network.
- Use the path determined in item 1 to balance traffic between multiple gateways to one network.
- Restrict use of particular routes by groups (a feature only Cray systems support).

Consider the network shown in Figure 11.

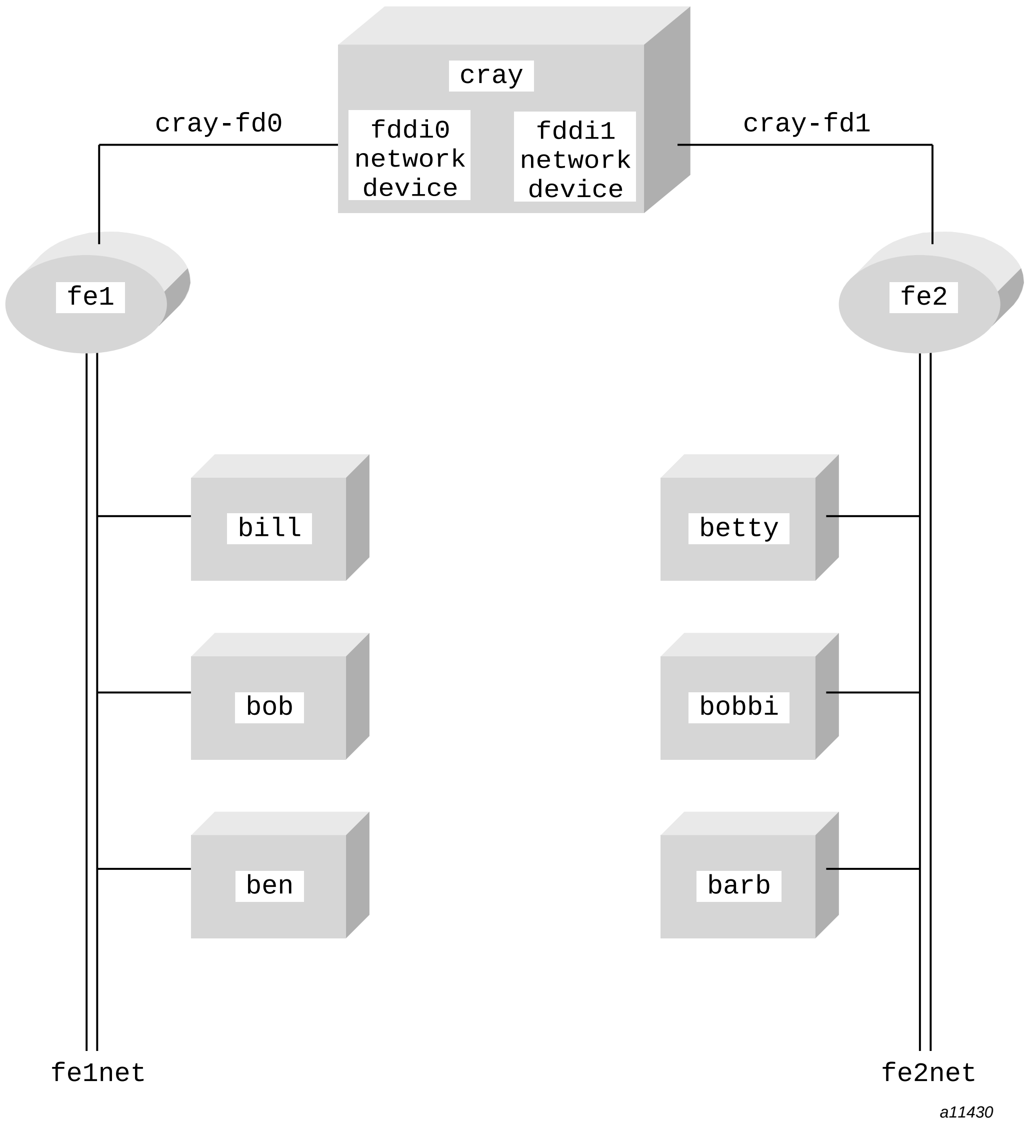


Figure 11. Network routing example configuration

Setting up this network is relatively simple. In your `/etc/tcpstart` script, you have `ifconfig`(8) commands to initialize network interfaces `np0` and `np1`. However, you must still add routes to the networks attached to each of the front-end systems. Therefore, the following route commands must be executed:

```
/etc/route add fe1net fe1
/etc/route add fe2net fe2
```

Of course, the routing tables on the front-end systems and the other hosts must be configured so that they can send datagrams to the Cray system.

2.3.3.1 Special Host Routing

You can use host-specific routes to cause data for a specific host to take a special route. Consider the example network shown in Figure 12.

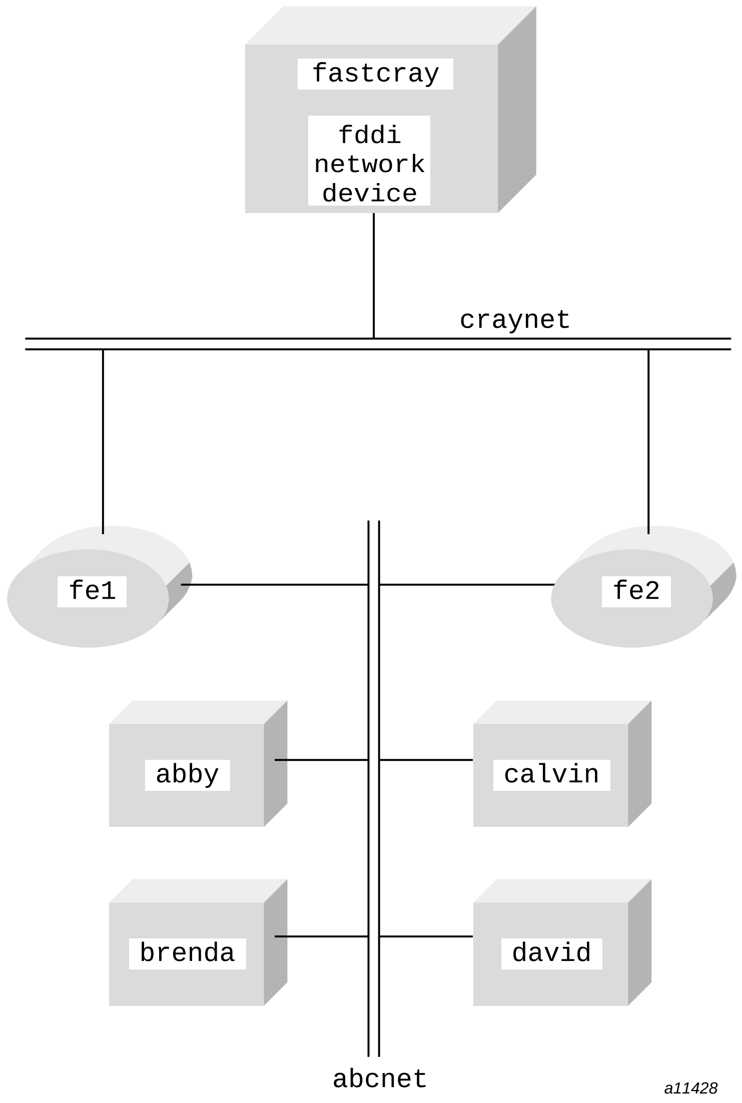


Figure 12. Special host routing example configuration

This network shows two front-end systems (`fe1` and `fe2`) connected to the same network, `abcnet`. By using a combination of host and network routes, it is possible to make most network traffic route through front-end system `fe1`, but you must allow for traffic between `fastcray` and host `calvin` to route through front-end system `fe2`.

The following `route` command routes traffic from `fastcray` to `abcnet` through `fe1`:

```
/etc/route add abcnet fe1
```

The following `route` command routes traffic from `fastcray` to `calvin` through `fe2`:

```
/etc/route add calvin fe2
```

The second route specification overrides the first because it is a host route, and host routes are chosen before network routes. However, the route to `calvin` does not affect routing to the other hosts on `abcnet`.

Of course, `calvin` is probably sending datagrams back to `fastcray`. If its default route is to `fe1`, data it returns to `fastcray` is routed through `fe1`. You can override this with either a network route to `craynet` or a direct host route to `fastcray`, specifying `fe2` as the gateway to use. `calvin` 's `route` command can be one of the following:

- To add a host route:

  ```
  /etc/route add fastcray fe2
  ```

- To add a network route:

  ```
  /etc/route add craynet fe2
  ```

Adding the network route implies that all traffic from `calvin` to network `craynet` routes through `fe2`. Adding a host route directs traffic bound for `fastcray` through `fe2`. It is best to use a host route for this situation, because if a network route is used and network data is sent to `fe1`, it must still be sent through `fe2`, rather than directly to `fe1`.

Special host routing like this can be desirable for both testing and performance reasons. However, even though network traffic for `calvin` is moved from `fe1` to `fe2` (thus reducing the network load that `fe1` must carry), all network traffic for `abcnet` still goes through interface `np0` on the `fastcray` system.

2.3.3.2 Load Balancing

If you want to balance the load on the `craynet` network, consider having two interfaces on `craynet`. You can then route the traffic over separate networks, balancing the load between them. Consider the example network shown in Figure 13.

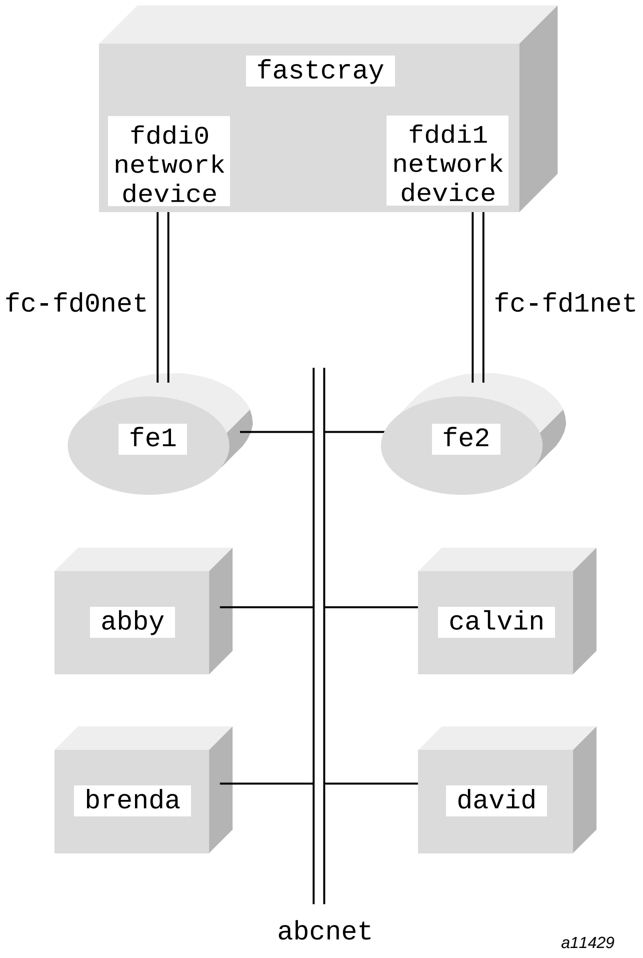


Figure 13. Load balancing example configuration

After using the `ifconfig`(8) command to initialize the interface, use the `route`(8) command to add the following routes:

```
/etc/route add abcnet fe1 (abcnet traffic through fe1)
/etc/route add calvin fe2 (calvin's traffic through fe2)
```

The `route` commands used to route traffic back from host `calvin` would be changed to one of the following to allow for the new network:

```
/etc/route add np1int fe2 (to add a host route)
/etc/route add fc-np1net fe2 (to add a network route)
```

It is best to use the host route.

Using this special host routing between gateways and networks can be an effective method of balancing network loads through front-end systems. You can also add special host routes for more than one host so that traffic for multiple special hosts is routed through specific gateways.

Of course, this balancing is limited by the bandwidth of the destination network. And even though the network load may be balanced between `fastcray` network interfaces and the front-end systems, the previous examples show all network traffic still traveling through `abcnet`, which may not have sufficient bandwidth to support it. In other examples, when the destination network has high bandwidth relative to the front-end systems, this form of load balancing can be very useful.

Having multiple interfaces on a Cray system, as shown in Figure 13, page 154, also reveals new routing problems. Front-end systems `fe1` and `fe2` have separate direct connections to `fastcray`. Therefore, all traffic from a front end to `fastcray` travels over the direct connection. However, this is not always true; datagrams travel over the connections on which they are routed.

Suppose that `fe1` is intended as the primary front end for the `fastcray` system, and the name `fastcray` is specified as the alias for `np0int`. This means that, when the name `fastcray` is resolved to an Internet address, the address of the `np0int` interface is returned. Routing decisions are then made based on this Internet address.

In this example, `fastcray` is the name with which most users are familiar. Therefore, a user on `calvin` might specify `fastcray` to access the `fastcray` system, and (because of the previous `route` commands) expect datagrams to travel from `calvin` to `fe2` and then to the `fastcray` system. However, because `fastcray` is an alias for `np0int`, datagrams route through `fe1` (if a route is set up to send default or `fc-np0net` traffic that way), or in the worst case, cannot connect.

Therefore, `calvin` must have routes to all of the `fastcray` system's interfaces through `fe2` to ensure that datagrams travel that path. You must add a route to `calvin` 's routing table for each of the `fastcray` system's network interfaces, as follows:

```
/etc/route add np0int fe2
/etc/route add np1int fe2
```

However, you must consider other issues. fe2 might not know anything about the fastcray system's np0 interface and either cannot route datagrams or sends them on to another gateway (fe1). Therefore, you must add a route to connect to fe2, as follows:

```
/etc/route add np0int np1int
```

This route sends fastcray traffic for np0int through the direct connection to np1int. It is apparent to the network software on the fastcray system that the incoming datagrams have reached their destination, and no more hops are necessary.

2.3.3.3 Controlling Routing by Group IDs

The route(8) command lets you specify particular groups that have permission (*inclusive* routing) or do not have permission (*exclusive* routing) to use a particular route; this is a special Cray system feature.

Groups are defined by the entries in the /etc/group file; see group(5). When a group list is specified for a route, the routing algorithm is changed slightly. In this example, when a route is selected, the routing algorithm checks to determine whether the user accessing the route is permitted to use it. If not, the routing algorithm continues looking for a route to the desired destination. Thus, it is possible to prevent some groups (and the users in those groups) from using particular routes, and to allow some groups to have access to special routes. This feature can also be used to prevent groups from accessing some networks and hosts completely.

Consider again the network shown in Figure 13, page 154, and the two route commands that follow it. If the second route command is changed as follows, only users in group specialgroup have network traffic from fastcray to calvin routed through fe2.

```
/etc/route add calvin fe2 -gid +specialgroup
```

This does not affect traffic from calvin to fastcray. Traffic from other groups travels from fastcray to calvin through fe1.

2.3.3.4 Controlling Access

With UNICOS security, the admin option of the ifconfig(8) command marks an interface as restricted to privileged users only. Creating a socket with PRIV_ADMIN effective on privilege-based systems, or with UID root on PRIV_SU systems enables a socket for communication on this interface. In a future release, an additional setsockopt system call will be required to enable communication on the socket. The `admin` option also prevents forwarding of IP packets to and from the interface.

2.3.3.5 Diagnosing and Fixing Routing Problems

When the TCP / IP network software is initialized, the routing tables are set up. Routes are created by the `ifconfig`(8), `route`(8), and `gated`(8) commands.

The `ifconfig` command creates a route only for the interface being configured and is not used to manipulate the routing table. The `gated` command can configure many routes from one configuration file and can dynamically change the routing tables. The `route` command is your primary tool for manually changing the routing tables. For more information on how to use these commands, see Section 2.2, page 22.

Diagnosing a routing problem requires checking the routing tables on all hosts involved to see how each host is routing datagrams between peers. Often, it is necessary for you to trace the route that is specified. You should be aware that routes are not symmetrical; that is, having a route between two hosts in one direction does not imply that there is a route back.

2.3.3.5.1 Using the `netstat`(8) Command to Inspect Routing Tables

You cannot tell from the routing table where a datagram will go after it has reached a gateway; therefore, you must inspect the routing table at the gateway to determine the next hop a datagram takes. You can inspect the routing table being used by the Cray system at any time by using the `-r` option of the `netstat`(1B) command. Options `-n`, `-s`, and `-v` can be used with `-r` to obtain more information about the routing table (see `netstat`(1B) for more information).

The `netstat -r` command prints out each routing table entry on one line. Host names are printed instead of Internet addresses, but you can request Internet dot addresses (such as `192.34.89.12`) by using the `netstat -rn` command.

The `netstat -r` command lists the routes in the order that the routing algorithm searches them. If the routes are not in a satisfactory order, use the `delete` and `add` options of the `route`(8) command to rearrange them. When the system is initialized, you should add the routes in the order desired. The output from `netstat -r` also tells you exactly how datagrams are being routed out of a host. Consider the following example output from `netstat -r`:

```
% netstat -r
Routing tables
```

```
Destination       Gateway             Flags    Refs   Use        Interface

Route Tree for Internet protocols
default           router-001          UG      63     103861      fd0
localhost         localhost           UH       9       1036      lo0
libe-ows          libe-ows-030        UGH      0          3      np0
vogonnet          fagin-ip-dev        UG       0         81      np1
torc-hyp          libe                U       12      20525      np1
torc-fddi         libe-fddi           U        8       1043      fd0
libe-030net       libe-030            U        6       6838      fd0
224.0.0.9         localhost           UH       0         10      lo0
%
```

Following is some of the information that this output provides:

- Destination `224.0.0.9` is a multicast address used by `gated`. It is an address for RIP Version 2 routers, `RIP2-ROUTERS.MCAST.NET`.
- Datagrams for host `libe-ows` are sent on interface `np0` in packets addressed to gateway `libe-ows-030`, because the route is a host route (host routes have an `H` in the `Flags` column) for host `libe-ows`.
- Datagrams bound for hosts on `vogonnet` travel through interface `np1` to gateway `fagin-ip-dev`. The `G` in the `Flags` column is set, which marks this as an indirect route.
- Datagrams destined for hosts on network `torc-fddi` are sent in packets directly to their destination, because the gateway flag is not set.

When inspecting the routing table, specifically notice the up flag (the `U` in the `Flags` column), because a route that is not up cannot be selected for use.

The `netstat -r` command (or its equivalent) is available on several systems other than Cray systems. Use the command that is appropriate for a particular computer to use to inspect that computer's routing table.

2.3.3.5.2 Tracing a Route between Two Hosts

Diagnosing a problem often requires tracing a route. The sample network in Figure 14 shows the route that datagrams take between two hosts. In this section the route is traced to point out why some hosts in the network cannot access a Cray system. This section also explains that an inefficient route is set up, and shows how to change it.

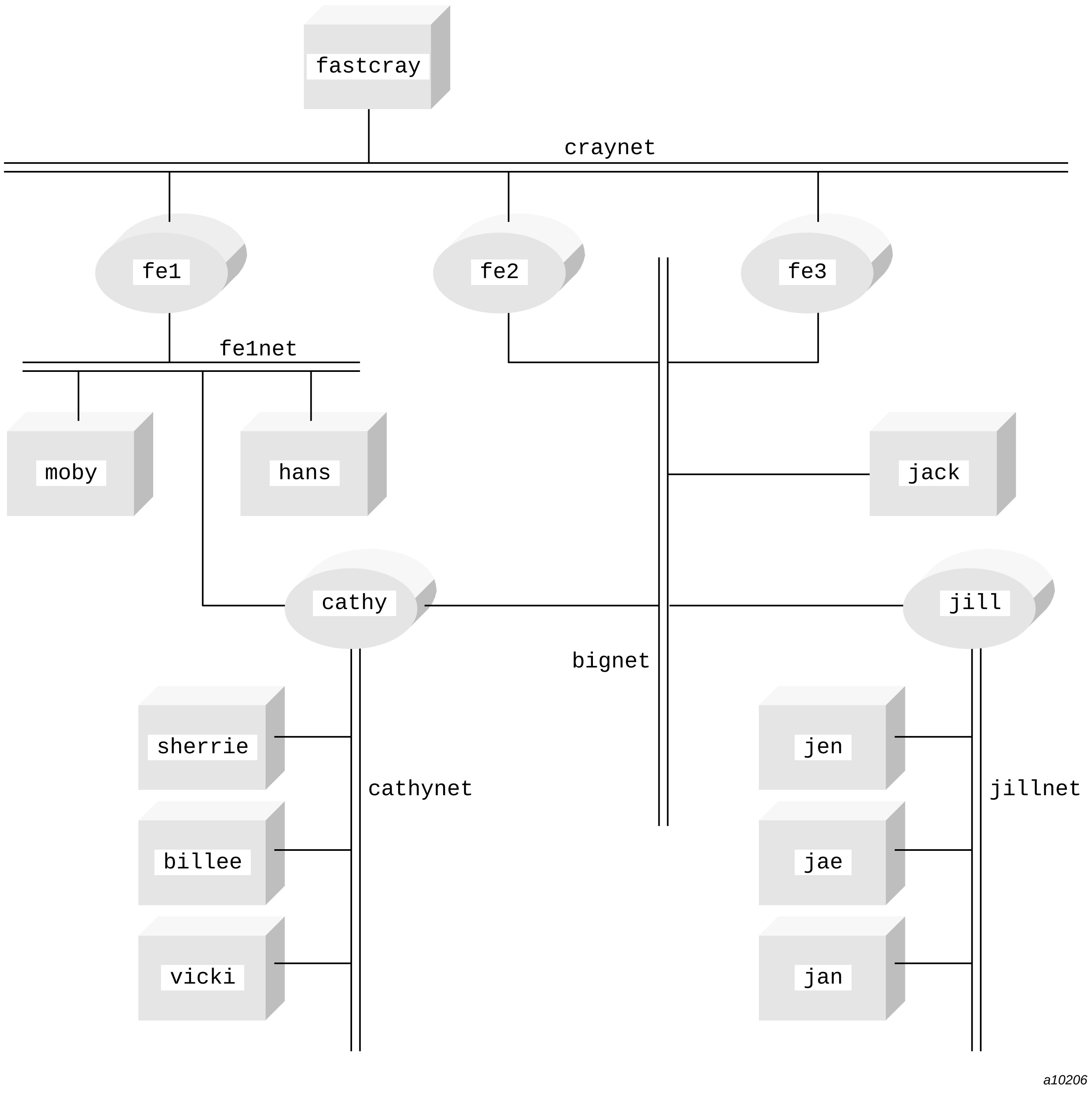
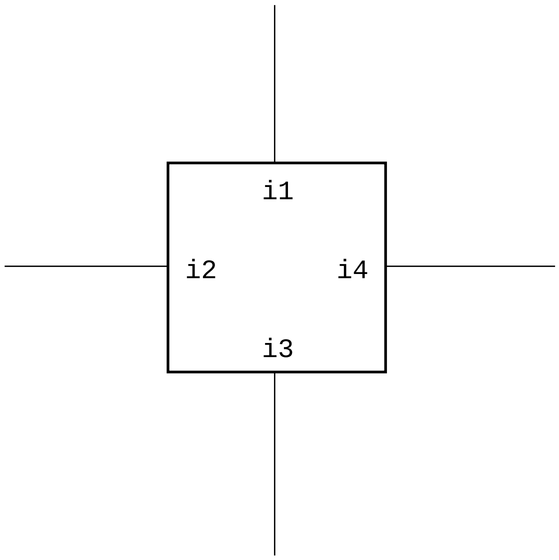


Figure 14. Route tracing example configuration

For simplicity, the following diagram does not show interface names. Assume the following conventions for interface names, relative to the position of the connection to a host's box:



a10259

Suppose that `jill` is a server, and `jen`, `jae`, and `jan` are workstation clients. A user at workstation `jae` wants to use `telnet`(1B) to access the Cray system `fastcray` and cannot get a connection (`telnet` prints a `Host unreachable` or `Network unreachable` error message). Because `jae` can `telnet` to `jack`, and `jack` can `telnet` to `fastcray`, all the network interfaces must be operating properly. Following the routes on each of the systems lets you determine why datagrams are not getting from `jae` to `fastcray`. First, check `jae` 's routing table.

```
# jae's routing table:

Destination     Gateway         Flags   Interface
localhost       localhost       UH      loop
jillnet         jae             U       i4
default         jill            UG      i4
```

From `jae` 's routing table you can see that datagrams for `fastcray` go to `jill`; therefore, you must now check `jill` 's routing table.

```
# jill's routing table:

Destination     Gateway         Flags   Interface
localhost       localhost       UH      loop
jillnet         jill            U       i3
bignet          jill            U       i2
default         cathy           UG      i2
```

From jill 's routing table, you can see why jae can communicate with jack. Because jack is on bignet, datagrams from jae to jack go through jill and then directly to jack. But there are no useful routes to fastcray; consequently, datagrams are sent to cathy over the default route. Therefore, you must go to cathy to see where the datagrams go from there.

```
# cathy's routing table:

Destination     Gateway         Flags   Interface
localhost       localhost       UH      loop
cathynet        cathy           U       i3
bignet          cathy           U       i4
fe1net          cathy           U       i2
jillnet         jill            UG      i4
craynet         fe1             UG      i2
```

cathy does not have a default route. Because cathy is connected to all networks in this internetwork, a default route is redundant. cathy is the default router for all the other computers in the network. cathy passes jae 's datagrams for fastcray on to fe1, because that is where cathy routes datagrams for hosts on craynet. Next, check fe1 's routing tables.

```
# fe1's routing table:

Destination     Gateway         Flags   Interface
localhost       localhost       UH      loop
fe1net          fe1             U       i3
craynet         fe1             U       i1
default         cathy           UG      i3
```

fe1 's routing table suggests that it forwards datagrams for fastcray directly to fastcray. Because you have traced datagram flow from jae to fastcray, you can be sure that datagrams can get from jae to fastcray. However, this does not ensure that datagrams can return to jae from fastcray. Therefore, the next step is to follow the path back, first by checking fastcray 's routing table.

```
# fastcray's routing table:

Destination     Gateway         Flags   Interface
localhost       localhost       UH      loop
craynet         fastcray        U       i3
fe1net          fe1             UG      i3
cathynet        fe1             UG      i3
bignet          fe3             UG      i3
default         fe2             UG      i3
```

As shown here, `fastcray` is connected to all networks except `jillnet`. Because `jae` is on `jillnet`, datagrams for `jae` are routed to `fe2`, the default route. Check `fe2` 's routing table.

```
# fe2's routing table:

Destination     Gateway         Flags   Interface
localhost       localhost       UH      loop
craynet         fe2             U       i1
bignet          fe2             U       i3
fe1net          fe1             UG      i1
cathynet        cathy           UG      i3
```

This routing table shows part of the problem. `fe2` appears to be configured as a master router for `fastcray`, but it does not have a route to `jillnet` or a default route. Consequently, when `fastcray` sends a packet for `jae` to `fe2`, `fe2` does not know how to get the packet to `jae`, and discards it. You can fix this problem by adding a route to `fe2` 's routing table; add either a default route to `cathy` or a network route to `jillnet`.

In this example, you can easily choose which route to add by considering that each hop a datagram takes lowers the overall performance over a network connection; that is, the fewer the hops, the better the performance. Therefore, the network route to `jillnet` should be added to `fe2` 's routing table.

However, a problem still exists with the routing tables. Datagrams from `jae` to `fastcray` travel through `cathy` and `fe1`, rather than taking the more efficient route through `fe2` or `fe3`. Performance is improved by adding a network route to `craynet` through `fe2` to `jill` 's routing table. `fe2` is chosen because datagrams from `fastcray` to `jillnet` are going the same route. However, using `fe3` as `jill` 's gateway to `craynet` also works. After these changes, `jill` 's and `fe2` 's routing tables contain the following:

```
# jill's routing table:
```

```
Destination     Gateway         Flags   Interface
localhost       localhost       UH      loop
jillnet         jill            U       i3
bignet          jill            U       i2
craynet         fe2             UG      i2
default         cathy           UG      i2

# fe2’s routing table:

Destination     Gateway         Flags   Interface
localhost       localhost       UH      loop
craynet         fe2             U       i1
bignet          fe2             U       i3
fe1net          fe1             UG      i1
cathynet        cathy           UG      i3
jillnet         jill            UG      i3
```

Now the routing tables are fixed so that `jill`’s workstation clients can communicate with the Cray system; an inefficiency in the routing tables is also fixed so that datagrams between `fastcray` and `jill`’s clients take the shortest path.

2.3.3.5.3 Using Traceroute to Trace Routes

The UNICOS system provides `traceroute`, a tool to help you trace routes without checking the routing tables on all systems involved. See `traceroute`(8) for syntax details.

The `traceroute` command is a useful for tracing the route datagrams take through a network; it is very useful for tracing routes over networks outside of your local administrative domain. However, `traceroute` relies on aspects of proper implementation of the IP protocol that do not exist in some environments; thus, it is not a complete substitute for tracing routes by examining the relevant tables.

If you were to use `traceroute` to trace the route from `fastcray` to `jae`, as shown in the sample network in Figure 14, page 159, before fixing the routes, you would see the following output:

```
# traceroute jae
traceroute to jae (132.162.155.14), 30 hops max, 56 byte packets
 1  fe2 (132.162.80.124)  11 ms  11 ms  11 ms (11.0/11.000)
 2  fe2 (132.162.80.124)  11 ms !N  11ms !N  11ms !N (11.0/11.000)
```

```
#
```

This output from `traceroute` shows that `fe2` cannot forward packets to `jae` (as indicated by the `!N` after the times for the probe packets). Quick inspection of `fe2`'s routing table shows immediately that `fe2` has neither a route to `jae`'s network (`jillnet`) nor a default route to try. After a route to `jillnet` is added to `fe2`'s routing table, the output of `traceroute` looks like the following:

```
# traceroute jae
traceroute to jae (132.162.155.14), 30 hops max, 56 byte packets
 1  fe2 (132.162.80.124)  11 ms  11 ms  11 ms (11.0/11.000)
 2  jill (132.162.90.17)  12 ms  13 ms  12 ms (12.0/12.334)
 3  jae  (132.162.91.8)  14 ms  14 ms  13 ms (14.0/13.667)
#
```

The `traceroute` command lets you trace routes from your Cray system to other computers, but if your other computers do not support `traceroute` or a similar facility, you cannot trace datagrams going to the Cray system. Then, manual inspection of routing tables is still necessary.

2.3.3.5.4 Using Snmproute to Trace Routes

The UNICOS system also provides `snmproute`, which is another tool to help you trace routes without checking the routing tables on all of the systems involved. This command uses the simple network management protocol (SNMP) to obtain the information (see RFC 1155). See `snmproute`(8) for syntax details.

The `snmproute` command is very useful for tracing the route that IP datagrams take between two nodes by using the Simple Network Management Protocol (SNMP). The `snmproute` command also provides other useful information about the interfaces.

The `snmproute` command performs a search of the routing table of the source node to determine the next IP node in the route. This process is repeated for each successive node in the route until the target node is found.

Thus, like `traceroute`, `snmproute` is not a complete alternative to examining routing tables. Unlike `traceroute`, however, `snmproute` can trace a route in both directions, providing a more complete view of the routing picture. `snmproute` also provides information about the network interfaces used to traverse the route.

For `snmproute` to determine a complete route to a target node, each node along the route must be running an SNMP agent listening at standard port 161. In addition, the SNMP agents must be configured to respond to the SNMP community name used by the snmproute command (the default community name is `public`). The `-c` option can be used to specify a different community name to be used by the `snmproute` command.

If intermediate hosts do not have an SNMP agent running, or if the community name is not known, `snmproute` provides routing information only up to the host which is not running the agent.

The following example assumes that SNMP agents are listening at port 161 and are configured to respond to community `public` for all of the nodes involved. If you were to use `snmproute` to trace the route from `hosta` to `hostb`, you would see the following output:

```
% snmproute hosta hostb
Tracing route from hosta(138.142.82.48) to hostb(138.142.82.27)

Out of 138.142.82.48 (hosta.mw.cray.com) I/F 18 (fd0) MTU 4352
Into 138.142.82.27 (hostb.mw.cray.com)
Reached 138.142.82.27 (hostb.mw.cray.com)
```

Here is an example in which a gateway between the source and target nodes is not configured to respond to queries for community `public`:

```
% snmproute hosta hostc
Tracing route from hosta(138.142.82.48) to hostc(138.142.21.90)

Out of 138.142.82.48 (hosta) I/F 18 (fd0) MTU 4352
Into 138.142.84.100 (router1)
No SNMP Response from router1
```

2.3.3.6 Labeling Route Entries with IP Type-of-Service (TOS)

Routes that you create with the `route`(8) command can be labeled with an IP Type-of-Service (TOS) value that can be used during routing lookups. When a routing lookup is performed in the kernel, the TOS that is assigned to the socket is passed as part of the lookup key. The look-up procedure attempts to find a route that is labeled with the exact TOS that is requested. If none is available, the procedure accepts the best fit. Of course, if there is no route for the particular destination, the lookup fails. If the route entry is marked for exact TOS matching (using the `-tosmatch` option on the `route`(8) command), every TOS bit that is set in the route must be set in the request.

TOS labels on routes can be used in a variety of ways. If your network has routers or bridges that can direct packets that are based on the TOS field in the IP header, you can use TOS labeling. You can also use TOS labeling to route packets to different gateways. For example, the `ftp` command can use TOS for its network connections. The command connection selects a low-delay bit in the IP TOS field by using the `delay` argument of the `-S` option; `ftp` data connections select the high-throughput bit in the IP TOS field by using the `throughput` argument of the `-S` option. Suppose you entered the following `route` commands:

```
route add bobo slowpoke
route add bobo quicky -S delay
route add bobo speedy -S throughput
```

The `ftp` command traffic to `bobo` goes to gateway `quicky` (`ftp` command connections use low-delay TOS), and `ftp` data traffic goes to gateway `speedy` (high-throughput TOS). The way in which the packets are routed beyond the first hop gateway depends on the remainder of the network and how it is configured. With this configuration, none of the FTP traffic is sent to `slowpoke`; however, other applications that do not use TOS use `slowpoke`.

You must also configure multiple route entries if you are using the `-tosmatch` option on the `route` command. For example, suppose your `route` entry has a TOS label, as follows:

```
route add soho mist -S delay
```

The only route to `soho` is through `mist`. All traffic for `soho` is sent to `mist`. However, suppose the `-tosmatch` option were specified on the `route` command line, as follows:

```
route add soho mist -S delay -tosmatch
```

Only sockets that have specifically requested low-delay TOS can send packets to `soho`. Other traffic needs another route entry to get to `soho`. The `-tosmatch` flag also means that sockets that specify high throughput cannot use a route that specifies low delay. However, a socket that specifies high throughput and low delay can use an exact match route entry for either high throughput or low delay (assuming that no route entry with both high throughput and low delay is available). Therefore, when adding a route entry with the `-tosmatch` option, it is advisable to add another route without a TOS entry, as follows:

```
route add soho mist -S delay -tosmatch
route add soho lake
```

If you are diagnosing a routing problem on a system in which TOS is being used, use the `-rv` option when running `netstat` so that you can see the TOS labels on routes. Using the `netstat` command with only the `-r` option does not display the TOS labels. You could be misled into thinking that a route is available when it is not because exact TOS matching is preventing some applications from using that route. Look in the flags column for the `T` flag when diagnosing this problem.

2.3.3.7 Preventing the Cray System from Becoming a Gateway

By forwarding packets, any host on the network can act as a gateway. Even when a host has only one network interface, routes can be configured so that the host receives datagrams to forward to another gateway. However, forwarding datagrams uses processing power; therefore, this is not an appropriate function for a Cray system. Other computers in the network should perform this function.

There are three ways to prevent your Cray system from becoming a gateway:

1. Ensure that none of the other computers in the network have gateway routes to the Cray system.

2. Mark some or all of the routes in the Cray system's routing table as being not forwarding. This is a feature specific to Cray systems. You can mark a route as being not forwarding by using the `noforward` option of the `route`(8) command when you add the route. Using the `-rv` option with the `netstat`(1B) command displays the `NOFORWARD` flag on a route if that route is marked not forwarding. If you choose this method, routes that are added by other commands (such as `ifconfig`(8)) must be deleted and then added again by using the `route` command.

3. Use the `netvar`(8) or the `sysctl`(8) command to turn off forwarding for all routes. These commands let you toggle the flag that activates forwarding (the `ipforwarding` flag) within the kernel. The `ipforwarding` flag is on by default, so that IP datagrams can be forwarded through the Cray system. The command to turn off forwarding is as follows:

   ```
   netvar -f off
   ```

   You can also use `netvar` in interactive mode to turn off IP datagram forwarding.

Note: When you are running the Cray ML-Safe configuration of the UNICOS system, the `ipforwarding` flag must be turned off. This is to prevent transmission of packets in violation of the MAC policy, which could happen because forwarding would be done at a layer below label processing.

You must be a super user to use `netvar` to change kernel variables. If you want to configure your Cray system so that forwarding is always turned off, you can use the `Configure System -> Network configuration -> TCP/IP configuration -> Kernel parameters` menu to change the `netvar.conf` file so that `netvar -f off` is executed at system startup.

When all routes on the Cray system are marked as being not forwarding, or forwarding is turned off with the `netvar` command, the Cray system cannot forward datagrams (a route that is marked as not forwarding is not selected for datagrams that need to be forwarded). If forwarding is attempted, an error is returned to the sender of the datagram, and the user sees this as a `Network unreachable` error. You must then trace the route the datagrams are taking, and fix the routing tables on the computers involved so that they are not trying to use the Cray system as a gateway.

2.4 Troubleshooting

Networking problems can be difficult to diagnose because of the number of individual elements that are involved in maintaining an entire network. Communication between the Cray system and an end user can be affected by problems in TCP/IP on the Cray system, hardware problems, bugs in the remote system's software, and bugs in other systems on the network. This section provides information to assist you in resolving these problems. It describes commands and tools, and it includes specific information about some of the daemons and network services.

This information assists you with your debugging problems, but it also is intended to help you test network changes before they are put into production. This information does not replace the information found in the appropriate man pages, but it only highlights what is available. Refer to the man page documentation for complete descriptions of the commands described in this section.

The following topics are covered:

- Troubleshooting tools
- Basic problem-solving strategy

- Network problems and solutions

  Note: RFC 1147 provides detailed information about monitoring and debugging TCP/IP networks.

2.4.1 Troubleshooting Tools

This section contains a description of the tools that are available to you for resolving network problems. The following categories of tools are discussed:

- Hardware diagnostics
- Network monitoring
- Network testing and diagnosing
- Network services

2.4.1.1 Hardware Diagnostics

Diagnostic tools should be used when you suspect that a network problem is caused by the actual networking hardware or UNICOS device drivers (including the IOS for Model E based systems, and MPN and HPN for GigaRing based systems). If a problem exists in any of these components, TCP/IP cannot communicate on the network. (But, TCP/IP's failure to communicate on the network does not always indicate a hardware problem.) These tests can be used to determine whether a problem is in TCP/IP, or in the hardware and its associated device drivers.

The commands that are listed in this section test and verify that the UNICOS driver (including the IOS driver for Model E based systems, and MPN and HPN for GigaRing based systems) and networking hardware are functioning properly. These diagnostic tools are a subset of all the tools available, and do not include the tools that are available with OLNET products. For a full description of all available diagnostic tools and the information they can provide, consult the appropriate hardware manual and software command documentation.

2.4.1.1.1 The `hit`(8) Command

The `hit`(8) command tests a Cray low-speed channel cabled in loopback mode, or an NSC A-series adapter and HYPERchannel connection.

Note: In the Cray ML-Safe configuration of the UNICOS system, special steps must be taken to run the `hit`(8) command (see *General UNICOS System Administration*, for more details on non-TCB software and UNICOS security).

2.4.1.1.2 The nx(8) Command

This command tests an NSC adapter (either A-series or N-series) and a HYPERchannel connection. The diagnostic can test communication over the HYPERchannel to the remote adapter.

2.4.1.1.3 The `scytest` Command

The `scytest` command tests a Cray system low-speed channel cabled in loopback mode, the FEI-3 interface cabled in loopback mode from the front end, VME-based system, or the FEI-3 interface between the Cray system and the front-end system. This diagnostic also verifies the front-end driver when testing the FEI-3 interface cabled in loopback mode, or when testing the interface between the Cray system and the front-end system. `scytest` is available through the supplier of the FEI-3 driver for your front-end, VME-based system.

For the fy driver, use the diagnostics that Cray supplies instead of this test.

Note: Cray J90 systems do not support the `scytest` command.

2.4.1.1.4 The `vht`(8) Command

The `vht`(8) command tests a HIPPI channel that is either in loopback mode or connected to another host.

Note: In ML-Safe UNICOS systems, special steps must be taken to run the `vht`(8) command (see *General UNICOS System Administration*, for more details on non-TCP software and UNICOS security).

2.4.1.2 Network Monitoring

You can use the commands listed in this section to determine the activities that are occurring over the network and in TCP/IP on the Cray system.

2.4.1.2.1 The `hyroute`(8) Command

The `hyroute`(8) command creates, displays, and verifies the information that TCP/IP is using to resolve hardware addresses. This display also includes the mtu value that TCP/IP is using for directly connected hosts.

2.4.1.2.2 The `arp`(8) Command

The `arp`(8) command creates, displays, and verifies the information that TCP/IP is using to resolve hardware addresses.

2.4.1.2.3 The `netstat`(1B) Command

The `netstat`(1B) command displays the contents of various network-related data structures. This information is valuable when determining the current status of the TCP/IP components of a given host. The following options are a subset of those available by using the `netstat`(1B) command. For a full description of the options, see the man page.

Option	Description
`-i`	Displays all of the available interfaces on a given host, and some specific information about them.
`-iv`	Displays the status of the queues for each interface, the number of packets discarded, and the flags.
`-is`	Displays statistics about the interfaces.
`-m`	Displays statistics associated with TCP/IP mbuf usage.
`-r`	Displays the routing information TCP/IP is using, in the order that the routes are being searched. See Section 2.3.3, page 148, for details about network routing.
`-A`	Displays information about all active connections.
`-a`	Displays information about all open sockets. This includes information about all available services on the host (the last column of the display indicates whether the socket is in the `LISTEN` state).
`-s`	Displays statistics associated with TCP/IP and packets received and sent.

Adding `-n` to any of these `netstat` options prevents `netstat` from translating numbers (that is, port numbers or Internet addresses) to their associated names.

2.4.1.2.4 The `nslookup`(1B) Command

The `nslookup`(1B) command provides an interface to the name server; `nslookup` displays the actual host name and Internet address that is associated with a specified name or address. This information is used by the network components that require the official host name or Internet address that is mapped to any given alias or address. This command interfaces only with the Berkeley Internet name domain (BIND) server. It is particularly useful if you are configuring and installing `named`(8) on a Cray system. By using `nslookup`, you can verify whether `named` is configured properly.

2.4.1.2.5 The `ping`(8) Command

The `ping`(8) command transfers a packet to a remote host and requests an echo. It is useful for verifying the connection between any two systems.

2.4.1.2.6 The `snmproute`(8) Command

The `snmproute`(8) command traces the route IP takes from the local system to the given remote host, and also the route back. `snmproute` can perform tracing on all hosts that are running SNMP (simple setwork management protocol) on the route between the local system and the remote host.

2.4.1.2.7 The `traceroute`(8) Command

This command traces the route IP uses when sending a packet from the local system to the given remote host. (It does not trace the route from the remote host to the local system.) `traceroute` continues until the remote host is reached. This command depends on aspects of IP that do not exist in every environment; therefore, it does not display hops through a system whose IP does not support this feature; instead, those hops are displayed as * (asterisks).

2.4.1.3 Network Testing and Diagnosing

The commands that are listed in this section provide information that is useful for testing and diagnosing TCP/IP problems.

2.4.1.3.1 The `nettest`(8) and `nettestd`(8) Commands

The `nettest`(8) and `nettestd`(8) commands perform client and server performance tests for the various types of network connections that the TCP/IP kernel code supports. This includes UNIX domain, TCP, and UDP socket connections.

The `nettest` command performs the client side of the test; the `nettestd` command performs the server side of the test.

Note: With the Cray ML-Safe configuration of the UNICOS system, special steps must be taken to run the `nettest`(8) and `nettestd`(8) commands (see *General UNICOS System Administration*, for more details on non-TCP software and UNICOS security).

2.4.1.3.2 The `trcollect`(8) and `trformat`(8) Commands

The `trcollect`(8) and `trformat`(8) commands collect and format, respectively, packets received and sent by TCP/IP on the Cray system. They accept options that determine which packets are traced. These commands provide information about network activities at the lowest possible level. See Section 2.5, page 206, for details.

2.4.1.4 Network Services

This section presents information about the following network services, which are available on the Cray system:

- `inetd`(8)
- `telnet`(1B)
- `ftp`(1B)
- `sendmail`(8)
- `gated`(8)

This information is useful when you are testing, debugging, planning to make changes to your system, or configuring a new service. These services are the ones most commonly used on Cray systems. For more information on these services, and other available services, see Section 2.2, page 22.

2.4.1.4.1 The `inetd` Daemon

The `inetd`(8) daemon listens for incoming requests on the configured ports and then starts up the appropriate program to process the request. As an administrator, you must add entries to the `/etc/inetd.conf` file to specify port numbers, commands to be executed, and so on. See Section 2.2.8.7, page 98, for a complete explanation of the file fields. If you do not have an `/etc/telnetd` entry in the file, and a user tries to use `telnet`(1B) to access the Cray system, `inetd` refuses the connection and issues a `Connection refused` message.

You can use the `/etc/inetd.conf` file to execute test versions of the various services before putting them into production. For example, adding the following line to the file lets you run another `telnet` daemon at port number `3200`:

```
3200  stream  tcp  nowait  root  /usr/src/tcp/usr/etc/telnetd  telnetd
```

Putting this line in the `/etc/inetd.conf` file lets you test out the latest `telnet` daemon before installing it into `/etc/telnetd`. That is, after upgrading to a new level of the UNICOS system, you could try using `telnet` to access the Cray system from the various front-end systems at your site, and verify that the new `telnet` daemon does not introduce any new problems. To use this `telnet` daemon, execute the following command on your front-end system:

```
telnet cray 3200
```

You can set up a test version of a service for any of the programs that are started by `inetd`. You can also use `inetd` to start your own special network services. `inetd` rereads its configuration file if you send it a `SIGHUP` signal (that is, if you execute `kill -HUP` *process_id*; *process_id* is the process ID of the running `inetd`).

2.4.1.4.2 The `telnet`(1B) Command

The `telnet`(1B) command provides user access to the Cray system. A wide variety of `telnet` implementations are used to gain access to the Cray system, and we try to accommodate as many variations as possible. As a result, the `telnet` daemon that is supported under the UNICOS system is sensitive to changes; that is, attempting to correct one problem can introduce other problems. This section assists you in resolving this type of problem.

When `telnet` is executed, the `telnet` client and `telnet` daemon negotiate the handling of both the data and the terminal. The user can monitor and manipulate this negotiating environment by using various `telnet` subcommands. The following subcommands are particularly useful for monitoring:

Subcommand	Description
`status`	Lets you obtain a printout of the current status of the `telnet` session, including whether `telnet` is in line-by-line or character-at-a-time mode.
`toggle` *options*	Lets you obtain, from the `telnet` client, a printout of all option negotiation that occurs between the `telnet` client and the remote daemon.
`toggle netdata`	Lets you obtain a printout, from the `telnet` client, of the hexadecimal representation of all data that is both sent to and received from the remote daemon.

All of the subcommands in the preceding list must be executed at the `telnet` prompt. To get this prompt, either execute `telnet` (rather than `telnet` *host_name*), or at any time during the logon session, enter the `telnet` escape sequence that is displayed during the connection sequence (usually ^]).

Note: The subcommands described in this section apply to the version of `telnet` that is running on Cray systems. For a list of the correct subcommands for the version of `telnet` that you are running, execute `help` at the `telnet` prompt.

Several subcommands are available to use to manipulate this environment. For example, the `mode character` command ensures that `telnet` operates in character-at-a-time mode (rather than line-by-line mode). See Section 2.2.8.7.6, page 104, for a description of the environment manipulation subcommands that are available.

When attempting to debug a `telnet` session, you must first isolate the failing component. Users sometimes try to access the Cray system through intermediate hosts. That is, rather than accessing the Cray system directly from their workstations, they access another host (or even several), and then try to access the Cray system. Attempting to access the Cray system through various combinations of hosts can help to isolate the failing component.

Typing the escape character returns the user to the original `telnet` environment. To monitor a second or even third `telnet` session, the user must first change the escape character in all previous sessions. This ensures that `^]` escapes to the `telnet` session you want to monitor.

2.4.1.4.3 The `ftp`(1B) Command

The `ftp`(1B) command provides user access to the Cray system. The following information describes methods for obtaining debug information from both the `ftp` and the `ftpd`(8) daemon when a problem is encountered.

Executing `ftp` puts the user in an environment that the `ftp` client sets up. In this environment, several subcommands are available. For example, `put` and `get` let the user send and retrieve files, respectively; `debug` lets the user see the commands that the `ftp` client is sending to the remote host.

The `ftp` protocol specifies which commands the client can send to the remote host, and also the meaning and format of the server's responses. Commands and responses are grouped in pairs. As a result, you can monitor the communication, and thus control the connection between the local client and the remote server.

You can obtain other information to assist in debugging `ftp` by specifying the following options to `ftpd`:

`-d` or `-v`	If these options are specified, `ftpd` logs debugging information to the system log, `auth`, defined in the `/etc/syslog.conf` file.

`-l`	If this option is specified, `ftpd` logs general information to the system log, `auth`, defined in the `/etc/syslog.conf` file.

Information in `syslog` is useful when determining the activities that are occurring on the server side of the connection.

2.4.1.4.4 The `sendmail`(8) Command

The `sendmail` program provides several modes of operation. One of the most useful modes is invoked by specifying the following:

```
/usr/lib/sendmail -bt
```

The `-bt` option lets you view the values that `sendmail` is using to interpret the configuration file (option `-C` *file* can be specified to instruct `sendmail` to use *file* as the configuration file). For an explanation of how `sendmail` interprets the configuration file, see the example on Section 2.4.3.2.2, page 200.

Another valuable `sendmail` mode is invoked by specifying the following:

`/usr/lib/sendmail -v -d`*x.y addressee*

Using this option rather than executing `mail` *addressee* lets you send email and print debug messages from the code in the numeric range *x* to *y*. For example, specifying `-d8.99` prints all debug messages that have a value greater than `8`. The output from this process assists you when determining the location of the problem.

2.4.1.4.5 The `gated`(8) Command

When `gated` is executed (typically by the `/etc/netstart` script), it can be given the argument of the name of a file to which it should log any tracing information during its run. For example, `gated` can be executed by using the following command line:

```
/etc/gated /usr/tmp/gated_log
```

This command causes `gated` to log information to the `/usr/tmp/gated_log` file, as directed by the `traceoptions` statement in the `/etc/gated.conf` file. The trace flags specified on the `traceoptions` statement specify the desired level of tracing output. At a minimum, the trace flags that are recommended are `general` and `mark`. See the `gated`(8) man page for a full listing of trace flags.

When `gated` is executed, logging of information to the file can be stopped and started dynamically by a signal sent to the running `gated` process. To make sending this signal convenient, `gdc`(8) is provided. `gdc` provides a user-oriented interface for the operation of `gated`.

The following signals have the identified effects on `gated`:

Signal	Description
`1 (SIGHUP)`	Causes `gated` to reread its configuration file to update specific pieces of information. See the `gated`(8) man page for details.
`2 (SIGINT)`	Causes `gated` to schedule a dump of its memory contents to the `/usr/tmp/gated_dump` file; this dump includes the status of all interface configurations and routes that are known to `gated`.
`49 (SIGUSR1)`	Causes `gated` to stop logging information (when currently logging information) or start logging information (when not currently logging information) to the log file that is specified when `gated` is executed.

2.4.2 Basic Problem-solving Strategy

This section describes a basic strategy for resolving network-related problems. This strategy can be used to isolate a failing network component. It also isolates the part of the network that is causing the problem. Because the ultimate resolution of any problem requires specific expertise in the failing area, this approach enables you to know whom to contact if you cannot resolve the problem yourself.

2.4.2.1 TCP/IP in a Cray System Environment

This section describes the manner in which TCP/IP components interact with each other; it does not describe how TCP/IP is configured on a Cray system. It is assumed that you are familiar with the following concepts:

- Addressing networks in a UNICOS environment (that is, the fact that a *network address* consists of the identification of a physical device and a logical path). See Section 2.1.1, page 5.
- Assigning Internet addresses for Cray systems. See Section 2.1.2, page 8.
- Mapping Internet addresses to their hardware equivalent. See Section 2.1.3, page 12.

- Determining logical paths on Cray systems. See Section 2.1.4, page 14.
- Enabling and disabling network interfaces. See Section 2.2.9, page 106.
- Specifying routing information. See Section 2.2.9.8, page 114.

Figure 15 illustrates the interaction of TCP/IP components that run on Cray systems.

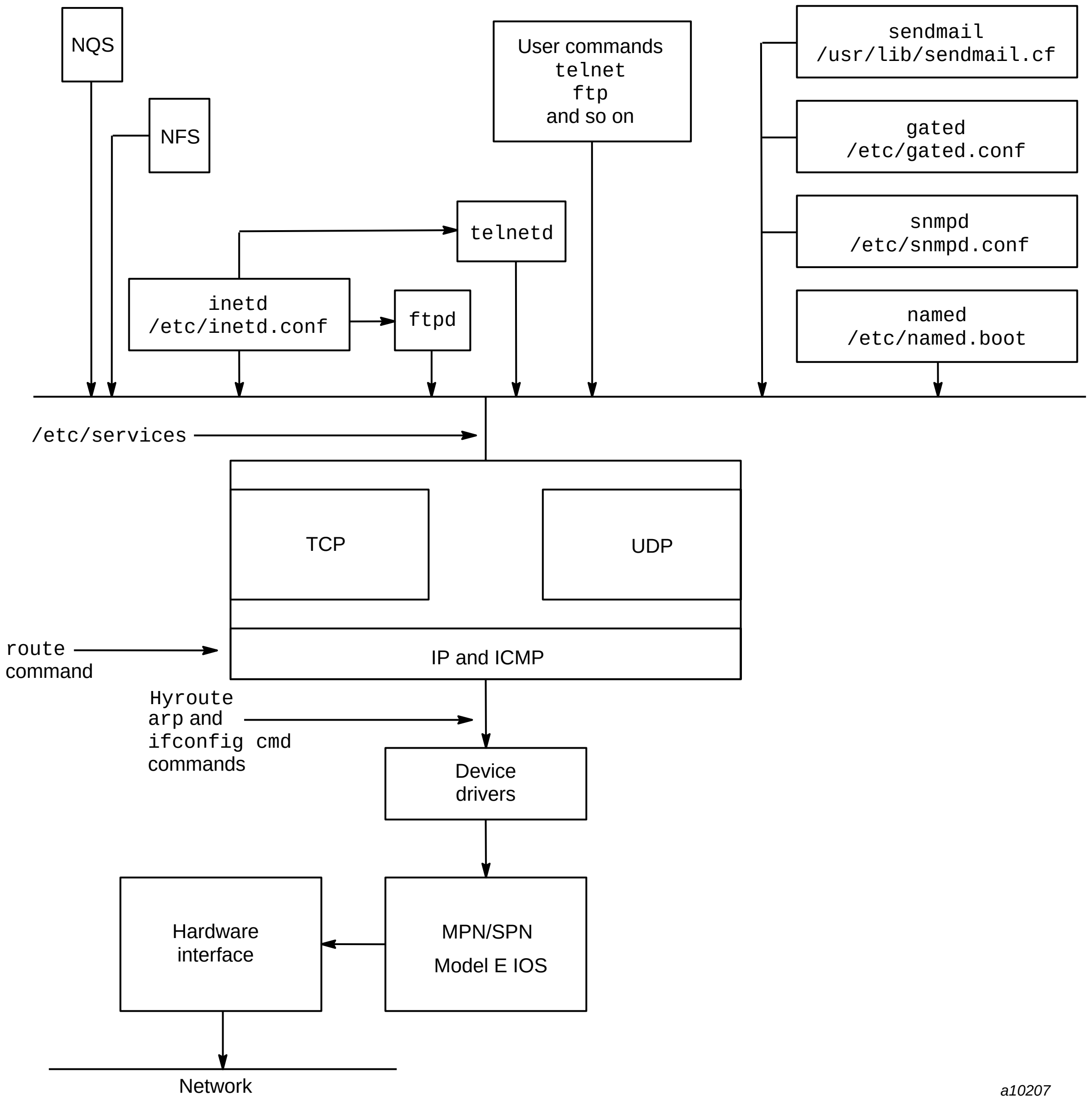


Figure 15. TCP/IP component interaction

Also shown in Figure 15, page 179 are the configuration files that some of the daemons use. For example, `inetd` reads in the `/etc/inetd.conf` file and uses this information to determine the network processes for which it listens.

The commands and files shown outside the boxes in Figure 15, page 179 indicate (with an arrow) which TCP/IP components or interfaces they affect. For example, the `/etc/services` file affects the interface between network applications and TCP/IP by using names to map applications to their port and network protocol.

Not identified on the chart are the `/etc/hosts` and `/etc/networks` files. These files map names (either host names or network names, respectively) to their Internet addresses. This mapping is used by all network components.

Note: When the `/etc/hosts.usenamed` file exists (its contents are meaningless), the mapping of names to Internet addresses is provided by the name server (that is, `named`) rather than the `/etc/hosts` file.

Assume that you are trying to determine why TCP/IP cannot communicate with any of the hosts connected to a certain hardware interface. The `ping`(8) command indicates that the remote hosts are not responding, and no other commands work by using this interface. Knowing that the `ping` command sends an Internet control message protocol (ICMP) `Echo` packet to another host, you can see in Figure 15, page 179 that at least one of the functions, which is located in the section from the bottom of the chart to the part of the chart that shows ICMP, is probably not working. That is, the problem exists in the hardware, device drivers, or the routing between the hosts.

Given the list of commands and components that are tested by the `ping` command, you can check to determine whether each is functioning properly. When `ping` indicates no response, you must check to ensure that `hyroute` and `ifconfig` commands are set up to configure the given interface correctly. The problem can range from the network interface not being enabled (that is, `ifconfig` was never executed to initialize the interface), to an error in the actual networking hardware.

After the problem is isolated to higher-level software, an increased amount of debugging information is available. Because each higher-level network service consists of a client program that is communicating with a server, another level of investigation is required to determine whether the problem exists on the client or the server side of the connection. The information provided in the documentation that accompanies the hardware can offer additional information to further isolate the problem.

For security issues, see Section 2.6, page 211.

2.4.2.2 Monitoring and Controlling System Changes

Because any change in a component on the network increases the probability that a problem will occur, it is important to monitor any network changes. It is also important to ensure, whenever possible, that any change made to any component on the network can be reversed. In this way, problems that are introduced by network changes can be resolved quickly.

For example, assume that users begin to encounter problems when they use `telnet` to access the Cray system. If you do not know what change occurred in the network, you must begin resolving the problem by trying to determine which component introduced the problem.

The task of resolving the problem can be simplified if changes can be reversed. Assume that you know that the `telnet` daemon on the Cray system was changed in the upgrade to a new bugfix release of the UNICOS system. If the previous version of the daemon was not deleted in the upgrade, you can replace the new version of the daemon with the old one and determine whether the problem disappears. If this corrects the situation, the appropriate analyst should be contacted to fix the current version of the daemon.

2.4.2.3 Isolating the Failing Component

When a network service is in use, several components are interacting with each other. These components can be grouped as follows:

- Components that comprise the networking software, such as routing, flow control, and so on
- Components that provide the physical connection, such as hardware and device drivers
- Components that provide the user interface, such as server and client programs

When attempting to isolate the network component that is causing a problem, you must first identify all components that are involved. For example, when a user uses `ftp` to transfer files between a front-end system and the Cray system, all of the networking code and hardware that exists between the front-end system and the Cray system, and also the client `ftp` program on the front end and the server `ftp` daemon on the Cray system, are involved. From this point, you can begin to use the various tools available to isolate the component that is causing the problem.

2.4.2.4 Isolating the Daemon and Client

To isolate a problem, you must first determine whether communications exist between the Cray system and the front-end system that has the problem. A very useful tool for determining this is `ping`(8). If `ping` is executed and shows that no packets are being lost, you know that data can be exchanged between the two systems, and that the problem must be in the higher-level software. If `ping` indicates that packets are being lost, data is not being exchanged between the two systems, and the problem is either in the networking software or in the actual hardware.

If data is being exchanged between the two systems, the problem is either in the client program or in the daemon program. One method of further isolating the problem is to try various combinations of client programs with the daemon program: the sequence of events that produces the problem should be tried from various front-end systems, because each front end has a different client program. Similarly, you can use one client program and vary the daemon programs. Then the sequence of events that produces the problem should be tried from one front-end system to varying remote systems.

See the client and server commands man pages for additional information.

2.4.2.5 Isolating the Hardware

Any problems that are encountered in the hardware and / or device drivers appear in data not being delivered between the Cray system and the host that is directly connected to the Cray system over the given physical hardware. The following tools are useful to determine whether the hardware is functioning properly between the Cray system and a directly connected host:

- The `ping`(8) command can be used to determine whether the Cray system can communicate over the physical hardware to the host to which it is directly connected.
- The `netstat`(1B) command with the `-i` or `-iv` option can be used to determine whether packets are being sent or received on the interface or whether errors are occurring.
- The various diagnostics that are available for the given type of hardware can be used.

The results from using these tools can be used to determine whether the problem is in the actual hardware or in the interface between TCP / IP and the device drivers. If the hardware diagnostics do not function properly (see the appropriate documentation on how to use the hardware diagnostics), the problem exists either in the device drivers or in the actual hardware itself. To further isolate the problem, you should consult an analyst who is familiar with the type of hardware you are using.

If the hardware diagnostics function properly, but `ping` indicates 100% packet loss, a problem exists in the interface between TCP/IP and the device drivers. By executing `ping` from both the Cray system and the directly connected host, and then reviewing the output from the `netstat` command, you can determine the following:

- Whether the Cray system is sending data that is not being received by the remote system
- Whether the remote system is sending data that is not being received by the Cray system
- Whether the Cray or the remote host is experiencing I/O errors when attempting to use the interface

The following items should be checked if you suspect the problem to be in the interface between TCP/IP and the hardware:

- Whether the Cray system is sending data to the correct hardware address of the remote host. This includes verifying that the Cray system is sending data to the correct logical path that TCP/IP has open on the remote system.
- Whether the remote system is sending data to the correct hardware address (including logical path) of TCP/IP on the Cray system.
- Whether the correct interface type is specified in the `ifconfig`(8) command.
- Whether valid mtu values are specified in the `hycf` file used by `hyroute`(8) when configuring the interface.
- Whether the interface in the IOS is initialized.

2.4.2.6 Isolating the Networking Software

If the Cray system can communicate with the hosts that are connected directly to its interfaces, but data is not being exchanged between the Cray system and other systems, the problem exists in the networking software on one of the hosts in the path between the Cray system and those remote systems. You must identify the host that is introducing the problem. The following tools are useful for determining whether the problem exists within TCP/IP on the Cray system or on another host:

- The `traceroute`(8) and `snmproute`(8) commands can be used to determine the route that packets are taking to allow the Cray system to communicate with the remote system.
- The `netstat`(1B) command can be used to determine the status of TCP/IP on the Cray system. Use this command to view various items such as mbuf usage, the current state of all active connections, and routing information.
- The `trcollect`(8) and `trformat`(8) commands can be used to trace packets going into and out of the Cray system.

Using these tools will help you determine where on the network the problem is occurring. Check the following items when you suspect that the problem is in the networking software:

- Whether the Cray system is running short of memory (that is, `netstat` with the `-m` option indicates that requests for memory are being denied).
- Whether a route exists from the Cray system to the remote host and from the remote host to the Cray system.
- Whether all hosts are up on the route between the remote system and the Cray system.
- Whether data, if any, is being exchanged between the remote system and the Cray system.
- Whether valid mtu values are specified on the `route`(8) command for this remote system.
- Whether the correct group restrictions are used on the `route` command for this remote system.
- Whether the interface is configured for privileged use only with UNICOS security (as in `ifconfig admin`).

2.4.3 Examples of Network Problems and Solutions

This section describes troubleshooting guidelines and shows several examples of networking problems and approaches you can take to resolve them. The solutions tell how the failing component is isolated and how the problem was ultimately resolved. These examples give you some ideas in resolving network-related problems; they do not describe a step-by-step method of resolving problems.

2.4.3.1 Troubleshooting Guidelines

Use the following guidelines to help you troubleshoot reported problems:

- For problems with installation and configuration involving the Cray L7R, see the *Cray L7R Release Overview and Software Installation Guide*.
- Understand exactly what the users are attempting to do and ask them for the exact syntax of their commands. For example, you should have the following information: Does the path involve a gateway? Are colons and periods in the right places? Are binary or ASCII files being transferred? Is case sensitivity an issue?
- Determine the remote host configuration. You must identify the operating system of the remote host and the networking product currently being run (because some products support a different set of protocols), identify the types of interfaces involved, and determine whether the remote host is the client or the server for the command.
- Determine whether the problem is specific to the local TCP/IP host or to the remote host. For example, if the user receives the error message `Connection refused`, are any of the following actions possible?
  - Can you perform a `telnet`(1B) operation back to the local TCP/IP host by using your Internet address?
  - Can you perform a `telnet` operation to any other remote host on the network?
  - Can anyone else perform a `telnet` operation to the problem host?
  - Can the remote host perform a `telnet` operation back to itself?
- Determine whether this is a recent or an ongoing problem.
- Determine whether changes were made to the system.
- Determine whether privilege is an issue; that is, is the command available only to the super user (or limited to certain categories with UNICOS security)?

The following are common problems that you might encounter:

- The remote or local interface is down.
- An incorrect address for the remote host exists in `/etc/hosts`.

- There is a conflict between the Internet address specified for a remote host in `/etc/hosts` and the address associated with it (as specified by the `ifconfig`(8) command).
- The `/etc/hosts` file has duplicate entries for the host name. Because this file is searched sequentially, the first entry is used.
- The user is not aware of case sensitivity associated with certain commands.
- Daemons are down.
- Routes were deleted or incorrectly entered with the `route`(8) commands.
- The gateway is down.
- The `/etc/hosts.bin` file is out of date.

2.4.3.2 Troubleshooting Examples

This section describes the most common problems that are encountered on the network and provides suggestions for solving them.

2.4.3.2.1 Connection Problems

The following problems can occur when a user is trying to connect to a remote host. In this section, the message is given (message text is shown in bold monospace font), followed by a description of the problem, the probable cause, and a possible solution. Some of the problems have more than one probable cause and possible solution listed.

> **Note:** There are several reasons for prohibiting connections with UNICOS security. See `spnet`(8) for an explanation of the NAL, WAL, and IPSO map tables and their effect on network connections. See `ifconfig`(8) regarding labels on interfaces and limiting the use of an interface to privileged processes. See "Restricting access to network interfaces," Section 2.6.3.4, page 219, for more details of these restrictions.

`Connection closed by remote host`

Problem:

A user has established a connection to a remote host, and the connection closes before the user logs out.

Cause:

This usually occurs because the remote host performed a shutdown and severed the connection, or the user process for the associated daemon was terminated.

Solution:

Use the `ping`(8) command to determine whether the host is running. Determine whether the remote host is down by contacting the host network administrator. If the host is not down, use `telnet`(1B) to reestablish the connection.

Connection refused

Problem:

Users cannot connect to a remote host. The remote host is up, and the interface is up, but a connection cannot be established.

Several possible causes and solutions for the problem follow.

Cause:

The wrong port number for the program is listed in the `/etc/services` file on the local or the remote host.

Solution:

Check the protocol's port number in the network services file (`/etc/services`) on the local host. It must agree with the port number shown in the default network services file (see an example of a network services file on page 36). If these numbers match, ask the network administrator on the remote host to perform the same check. Assume that the result of the check is as follows:

```
&Network services, Internet style
#
telnet   25/tcp
```

The `telnet` daemon should have port number `23` instead of `25`; therefore, the `/etc/services` file that is in error must be edited accordingly. The `/etc/services` data file should always keep the standard protocol port numbers. When the client program is started (that is, a user initiates a command), it opens `/etc/services`, obtains a port number, and looks for a daemon program that is listening on that remote host port.

When the daemon program is started (at boot time or when the network administrator starts it separately), the daemon opens `/etc/services`, obtains a port number, and listens for a request on that port. Therefore, if someone changes the port number in `/etc/services` on the client, the client will request a port number other than the one on which the standard daemon program is listening.

If the daemon program is listening on a nonstandard port number, it does not recognize a client's request on a standard port number. A connection cannot be established unless both port numbers agree.

If it is necessary to use a nonstandard port number (this is often done for testing), use the following procedure to start a daemon or client program on a different port number without interfering with the standard network programs and daemons:

1. To start a `telnet` server on port `35`, add another line to the `/etc/inetd.conf` file, as follows:

   ```
   35 stream tcp nowait root /etc/telnetd telnetd
   ```

   This line enables two `telnet` servers, one at port `23` (default) and the other at port `35`. (You can test a trial version of `telnet` by enabling two `telnet` servers.)

2. To connect to the `telnet` server on port `35`, the client `telnet`(1) program from a remote host issues the following commands:

   ```
   # telnet
   telnet> open hostname 35
   ```

Now the two servers are talking because the port numbers agree. (If a server is not invoked by `inetd` (for example `lpd` or `sendmail`), these commands can be entered directly from the command line.)

In an implementer's own programs, the port number is sometimes embedded in the code. If this is the case, check the source code.

For information about the syntax and format of the `/etc/inetd.conf` file, see Section 2.2.8.7, page 98.

Cause:

The `/etc/inetd` daemon is down.

Solution:

If TCP/IP users cannot connect to any of the networking services on a particular host, contact the network administrator for the remote host to determine whether `inetd` is operating.

If remote users cannot connect to the local TCP/IP host, enter the `ps`(1) command with the `-ae` options to determine whether `inetd` is running. For example, the following `ps -ae` output indicates that `inetd` is not running, because `inetd` is not listed:

```
# ps -ae

81   ?   0:00     lpd
83   ?   0:00     sendmail
```

The following dialog activates `inetd` on the TCP/IP host and verifies that the daemon is now running:

```
# /etc/inetd

# ps -ae

 81   ?   0:00     lpd
 83   ?   0:00     sendmail
 85   ?   0:00     inetd
```

Cause:

The server is not enabled by `/etc/inetd.conf`.

Solution:

If TCP/IP users can connect with other network services but cannot connect with a particular service on a host, contact the network administrator for the remote host to determine whether the daemons on the host are operating.

If remote users cannot connect to a particular service, but can connect to other services on the local TCP/IP host, your `/etc/inetd.conf` file is probably not set up correctly. Enter the `netstat` command with the `-a` option to determine whether the remote service is running.

The following output from `netstat -a` verifies that the `telnetd` service is not listening because `telnet` is not listed:

```
# netstat -a

Active connections (including servers)
Proto   Recv-Q   Send-Q   Local Address   Foreign Address   (state)
tcp       0        0      *.smtp          *.*               LISTEN
tcp       0        0      *.exec          *.*               LISTEN
tcp       0        0      *.shell         *.*               LISTEN
tcp       0        0      *.login         *.*               LISTEN
tcp       0        0      *.finger        *.*               LISTEN
tcp       0        0      *.ftp           *.*               LISTEN
```

If the `telnetd` service is not listening, view the `/etc/inetd.conf` file to determine whether the `telnet` service is enabled. If not, add the `telnet` entry to the `/etc/inetd.conf` file and issue a `SIGHUP` signal to the `inetd` process.

Use `netstat -a` again to verify that the daemon is now running:

```
# netstat -a

Active connections (including servers)
Proto   Recv-Q   Send-Q   Local Address   Foreign Address   (state)
tcp       0        0      *.smtp          *.*               LISTEN
tcp       0        0      *.exec          *.*               LISTEN
tcp       0        0      *.shell         *.*               LISTEN
tcp       0        0      *.login         *.*               LISTEN
tcp       0        0      *.finger        *.*               LISTEN
tcp       0        0      *.ftp           *.*               LISTEN
tcp       0        0      *.telnet        *.*               LISTEN
```

Connection timed out

Problem:

The local host is trying to connect to the remote host but is not making an active connection. The `telnet` program sent a packet to a specific address, but no hosts are responding. The `telnet`(1B) client program times out if a connection is not made within a certain time limit (usually between 30 and 75 seconds).

Example:

Assume that the user from the local TCP/IP host issued the following command:

```
# telnet hostname
telnet ....... connection timed out
```

Several possible causes and solutions for the problem follow.

Cause:

The remote host or network is down.

Solution:

If you suspect that the remote host or network is down, contact the network administrator for the host and request the host's status.

Cause:

With UNICOS security, there is not a NAL entry that authorizes access by your host or at your security label. The remote host appears to be down.

Solution:

If you suspect that the remote host or network is down, contact the network administrator for the host and request the host's status.

Cause:

An incorrect Internet address is specified for the remote host.

Solution:

A conflict in the Internet address can arise from the following situations:

- A user entered an incorrect Internet address on the command line. The user should retry the command, using the official host name or alias (to avoid reentering an incorrect Internet address).
- A network administrator changed the Internet address of the remote host without informing the other hosts on the network. The address associated with the host is different from that in the local `/etc/hosts` table. The network administrator for the host should verify that the address is correct.
- The local host table contains varying entries for the remote host. Only the first entry is used, regardless of which one is correct. Therefore, if multiple entries for a remote host appear in your `/etc/hosts` file, ensure that the first one is correct and delete the others.

Cause:

There is a hardware problem.

Solution:

Check the connections between the Cray system and the network hardware. (Use the `hit`(8) command to determine whether the network hardware is working.)

> **Note:** In the Cray ML-Safe configuration of the UNICOS system, special steps must be taken to run the `hit`(8) command (see *General UNICOS System Administration*, for more details on non-TCP software and with UNICOS security).

Cause:

The remote host's interface is down. (If the network hardware interface on the local TCP/IP host is down, the error message `Network is unreachable` appears.)

Solution:

If you suspect that the remote host's network interface is down, contact the network administrator and request the host's status.

Cause:

Nonexistent or incorrect routes are set up.

Solution:

If you suspect that the remote host's `route`(8) commands are configured incorrectly, contact the network administrator and request the host's status. If the `route` entry on the local TCP/IP host is at fault, the error message `Network is unreachable` appears.

`Login incorrect`

Problem:

A user is attempting to issue the `rlogin`(1B) command and receives the error message `Login incorrect`.

Cause:

The error message is returned from the remote host because the user entered an invalid user name or password for the host. When an invalid user name is specified, or an autologin is impossible because `/etc/hosts.equiv` or `$HOME/.rhosts` is not set up to allow an autologin, the user receives a prompt for a login and password.

Solutions:

- Set up the `/etc/hosts.equiv` or `$HOME/.rhosts` file correctly so that the user is not prompted for a password. See Section 2.2.5.3, page 37, and Section 2.2.11.2, page 121, for more information.
- Users must know their valid accounts and passwords on the remote hosts.

`The .netrc file not correct mode`

Problem:

A user attempts to use `ftp`(1B) with autologin to a remote host. The user's `$HOME/.netrc` file is set up properly with *machine*, *username*, and *password* entries, but the error message is returned, and `ftp` aborts.

Cause:

The file has read permissions set for someone other than the owner. Because of the sensitivity of information in `$HOME/.netrc`, the `$HOME/.netrc` file should be accessible only to the owner.

Example:

The following permissions are incorrect:

```
-rw-r--r--   1 peter other         24 Dec 26 14:14 /usr/peter/.netrc
```

Solution:

Change the access mode of the `.netrc` file to `600`, as follows:

Example:

```
#  chmod 600 /u2/peter/.netrc
```

The following dialog verifies that the mode was changed:

```
# ls -la .netrc
-rw-------   1 peter other         24 Dec 26 14:14 /usr/peter/.netrc
```

`Network is unreachable`

Problem:

The user attempts to connect to a host outside the local area network (LAN) and cannot establish the connection. Both hosts are known to be up.

Cause:

- The local network hardware interface is down. If the remote interface is down, the error message `Connection refused` is returned.
- The local network hardware interface is up, but the Internet address associated with it is incorrect. The interface is on a network that is different from the one for which it was intended.
- The route was deleted for the local interface.
- A route was not set up for the destination host.

- An incorrect route was set up for the destination host.
- The configuration file specified for `/etc/hyroute` does not properly map Internet addresses to hosts, or it does not properly list gateways to other low-speed channels.

Solution:

1. If users cannot perform a `telnet`(1B) operation back to themselves (using the local host name rather than loopback), the problem is in the interface.

   Example:

   User `peter` on `crayhost` issues the following command:

   ```
   # telnet crayhost
   Trying.....
   network is unreachable
   ```

   Enter the `netstat` command with the `-i` option to determine whether the local interface board is up, as follows:

```
# netstat -i

Name   Mtu   Network   Address     Ipkts  Ierrs  Opkts  Oerrs  Collis
np0*   1500  twgnet    crayhost    59803  0      45458  0      0
lo0    1536  Loopback  localhost   5601   0      5601   0      0
```

   The asterisk following `np0` indicates that the interface is down. Use the `ifconfig`(8) command to initialize the interface, as follows:

   ```
   ifconfig np0 crayhost up
   ```

   The following output from `netstat -i` verifies that the interface is up:

```
#  netstat -i

Name   Mtu   Network   Address     Ipkts  Ierrs  Opkts  Oerrs  Collis
np0    1500  twgnet    crayhost    59803  0      45458  0      0
lo0    1536  Loopback  localhost   5601   0      5601   0      0
```

   Try again to perform a `telnet` operation back to the local host. If you are unsuccessful, check the `hyroute`(8) command in `/etc/netstart`. The `hyroute` command should always be run before `ifconfig`.

   Determine the name of the configuration file that is specified on the `hyroute` command line and examine the file. Ensure that the correct network hardware address for the user's host interface is properly mapped to the user's host name. Make any necessary corrections and try again to perform a `telnet`(1B) operation back to the local host.

2. If users can perform a `telnet`(1B) operation to themselves, but not to anyone else on the local network, verify the address of the local interface.

   Example:

   User `peter` is on `crayhost` (local network `32`) and is trying to perform a `telnet` operation to `cray2`, as follows:

   ```
   #  telnet cray2
   Trying ...
   network is unreachable
   ```

   Enter the `netstat -ni` command:

```
# netstat -ni

Name    Mtu   Network    Address      Ipkts  Ierrs  Opkts  Oerrs  Collis
np0     1500  31         31.0.18.121 1373    0      753    0      0
lo0     4608  127        127.0.0.1   394     0      394    0      0
```

   Although `peter` thought his network hardware interface was configured on network `32`, it was actually configured on network `31`. Therefore, `peter` must perform the following steps:

   First check the Internet address in the `/etc/hosts` table for the interface. The following dialog shows that `crayhost` was set to an incorrect address:

   ```
   #  cat /etc/hosts

   31.0.18.121   crayhost
   32.0.0.2      cray2
   ```

   Change the Internet address in this host file, and reassign the new address to the interface, using the following `ifconfig`(8) command (first ensure that you change the Internet address in the `/etc/hosts` file). Verify by using `netstat -ni`. The dialog follows:

```
#  ifconfig np0 down
#  ifconfig np0 crayhost up

Name    Mtu   Network     Address      Ipkts   Ierrs   Opkts   Oerrs  Collis
np0     1500  32          32.0.18.121 1373     0       753     0      0
```

```
lo0    4608  127         127.0.0.1   394     0       394     0      0
```

If the host table appears to be correct, check the `ifconfig` command in `/etc/netstart`. This command should use the official host name instead of the Internet address.

A common error occurs when users add their official host name to the loopback line (`127.0.0.1`). When they initialize the network, their interface board is assigned to `127.0.0.1` rather than to the intended Internet address in `/etc/hosts`.

Try again to perform a `telnet`(1B) operation to a host on the local network.

3. If users can perform a `telnet`(1B) operation to themselves and to anyone on the local network, but not to a host on another network, the route is the problem.

   For example, `peter` is on network `32` (local host `crayhost`) and wants to perform a `telnet` operation to `cray3` on network `68`.

   Enter `netstat -nr` to verify the following routes that are set up, as follows:

```
# netstat -nr

Routing tables
Destination     Gateway       Flags   Refs     Use    Interface
32              32.0.18.121   UG      4        1129   np1
127             127.0.0.1     U       0        0      lo0
```

   This route table shows that `peter` can get to only networks `32` and `127` (loopback). `peter` must set up a route for network `68`. For example, if host `kit` on his network were the gateway to network `68`, `peter` would add the route as shown in the following example.

```
# route add net 68 kit
# netstat -nr

Routing tables
Destination     Gateway     Flags   Refs     Use   Interface
68              32.0.0.4    UG      0        354   np0
32              32.0.18.121 UG      4        1129  np0
127             127.0.0.1   U       0        0     lo0
```

   Because `kit` is the gateway to network `68`, it contains two addresses: one for network `68`, and one for local network `32`. Because `crayhost` is on network

32, the gateway host address that should be used is the same as that of crayhost (for example, 32.0.0.4).

When multiple gateways are involved, another problem occurs with routes. Then you must determine which intermediate gateway is set up with improper routes.

If the fault does not lie with the routing commands, determine the name of the configuration file invoked by the hyroute(8) command, and examine the file. Ensure that the remote host has a gateway configured for it, and that the correct hardware address for the gateway has been properly mapped to its host name. Make any necessary corrections and try again to perform a telnet(1B) operation to the remote host.

Ruserok: permission denied

Problem:

A user is trying to use the rsh(1), remsh(1B), rcp(1), or rlogin(1B) command and receives the error message.

Example:

User peter on crayhost attempts a remote shell (rsh or remsh) command on a given host through sue 's account, and issues the following command:

```
$ rsh hostname -l sue who
ruserok:  permission denied
```

Several possible causes and solutions follow:

Cause:

The $HOME/.rhosts file on the remote host does not exist or is not set up to allow access to the user.

Example:

User sue's $HOME/.rhosts file is not set up to allow peter to use her account; the contents of her $HOME/.rhosts file are as follows:

```
crayhost sally george
```

In this example, sue has authorized access for sally and george, but not for peter.

Solution:

Create or update the `$HOME/.rhosts` file on the remote host to allow access.

Example:

User `sue` 's `$HOME/.rhosts` file should be modified as follows:

```
crayhost sally george peter
```

Cause:

The `$HOME/.rhosts` file on the remote host is not owned by the user who owns `$HOME`.

Example:

User `sue` 's `$HOME/.rhosts` file is owned by `anne`, as shown in the following `ls`(1) output:

```
$ ls -l .rhosts
-rw-------   1 anne     users         22 Dec 16 15:09 .rhosts
```

Solution:

The super user must change the owner of the `$HOME/.rhosts` file to the owner of `$HOME`.

Example:

The super user uses the `chown`(1) utility to change the file's ownership, as follows:

```
# chown sue .rhosts
# ls -l .rhosts
-rw-------   1 sue      users         22 Dec 16 15:09 .rhosts
```

Cause:

The host name is not specified.

Example:

Host `birch` has two names corresponding to two IP addresses. The `$HOME/.rhosts` file should include both names to ensure a name that matches the address used is found.

Cause:

The `$HOME/.rhosts` file on the remote host is publicly writable.

Example:

Other users can modify user `sue`'s `$HOME/.rhosts` file, as the following `ls` output shows:

```
$ ls -l .rhosts
-rw-rw-rw-   1 sue      users         22 Dec 16 15:09 .rhosts
```

Solution:

Change the permissions on the `$HOME/.rhosts` file to `600`.

Example:

Use the `chmod`(1) utility to change the file's permissions, as follows:

```
$ chmod 600 .rhosts
$ ls -l .rhosts
-rw-------   1 sue      users         22 Dec 16 15:09 .rhosts
```

Unknown host

Problem:

A user tries to connect to another host and receives the error message. This error is generated from the client side.

Example:

```
# ftp otherhost
hostname: Unknown host.
ftp>
```

Cause:

There is an invalid entry in `/etc/hosts`. Each host to which the user wants to connect must have a valid entry in the local `hosts` file unless the user specifies the Internet address of the destination host. The remote host does not need an entry for the user in the `hosts` file, because the IP header contains the source address.

Solution:

Check the entry in `/etc/hosts` for the host name to which the user is trying to connect. Ensure that no comment indicator (#) precedes the entry.

Problem starting up /etc/netperf

Problem:

The user cannot start `netperf`(8).

Several possible causes and solutions follow.

Cause:

The display workstation does not support the X Window System (X) server; consequently, the following message is displayed, indicating that the X server is not enabled:

```
Connection refused
```

Solution:

You must run `netperf` on a bit map display workstation that supports X.

Cause:

The setting of the `DISPLAY` environment variable (after the user logs on to the Cray system) does not specify the displaying workstation.

Solution:

Set the `DISPLAY` environment variable, as follows:

```
setenv DISPLAY myworkstation:0
```

Ensure that the displaying workstation has the Cray system in its `Xhosts` list, which is a list of the hosts that can display X output.

2.4.3.2.2 Sendmail Problems

This section discusses error messages that can occur when a user is trying to use the `sendmail` facility. It offers guidelines for analyzing and resolving the error message problems.

`Host unknown`

A user attempts to send email from the Cray system, and the email is returned as undeliverable. This problem started only recently. The following is an example of a message returned by the system:

```
>From root Thu May 24 07:45 GMT 1990
Return-Path: <MAILER-DAEMON>
Received:  by myhost.domain.name
        id AA07406; 5.61/CRI-7.0; Thu, 24 May 90 02:45:18 -0500
```

```
Date: Thu, 24 May 90 02:45:18 -0500
From: Mail Delivery Subsystem <MAILER-DAEMON>
Full-Name: Mail Delivery Subsystem
Message-Id: <9005240745.AA07406@myhost.domain.name>
Subject: Returned mail: Host unknown
To: yourid

   ----- Transcript of session follows -----
550 yourid@yourhost... Host unknown

   ----- Unsent message follows -----
Received:  by myhost.domain.name
        id AA07404; 5.61/CRI-7.0; Thu, 24 May 90 02:45:18 -0500
Date: Thu, 24 May 90 02:45:18 -0500
From: John Doe <yourid>
Full-Name: John Doe
Message-Id: <9005240745.AA07404@myhost.domain.name>
To: yourid@yourhost

This is the message I wanted to send to you.
```

The `Host unknown` message indicates that `sendmail` could not find the name `yourhost` in the table of host names. This table provides the mapping of names to Internet addresses.

Using the preceding information, you can conclude that one of the following problems exists:

- The `sendmail` program contains an error.
- The `sendmail.cf` file is not configured properly.
- The routine that performs the host mapping function (`/etc/hosts` or the name server, depending on which one your site is using) is not correct, which causes the output that is returned by `gethostbyname`(3) to be in error.

The reason for this conclusion is that these are the only components involved in determining valid host names. The part of the `sendmail` program that creates destination addresses is suspected because the destination address cannot be found. The process of resolving valid host names is suspected because the error message indicates that the host name is unknown.

To isolate the cause of this problem, look at the changes that are made to these components on the Cray system. For this example, assume that you have upgraded the version of TCP/IP that is running, you have changed from using the `/etc/hosts` file to using the name server, and you have changed the `sendmail` configuration file.

First, check to determine whether the problem was introduced by the change from the `/etc/hosts` file to the name server. To test this possibility, you must go back to using `/etc/hosts`. Because the existence of the `/etc/hosts.usenamed` file indicates that the name server is being used, delete this file.

Next, attempt to send a test message to `yourid@yourhost`. If the email is delivered to `yourid@yourhost`, the `/etc/hosts` file is not in error.

Using this information, you can conclude that the problem is either with the method that `sendmail` used to interface with the name server or with the name server itself. Change back to the name server, and look at the tools available to assist in debugging the name server. The program `nslookup`(1) is a tool that queries the name server and displays the results. Following is a sample `nslookup` dialog:

```
prompt> nslookup
Default Server:  myhost.domain.name
Address:  128.162.75.1

> yourhost
Server:  myhost.domain.name
Address:  128.162.75.1

Name:    yourhost.domain.name
Address:  128.162.154.12

> ^D
prompt>
```

The output from `nslookup` indicates that the name server does know about host `yourhost`. Therefore, the problem must be with the method that `sendmail` used to interface with the name server. To determine the host name that `sendmail` is using when communicating with the name server, execute the following:

```
prompt> /usr/lib/sendmail -bt
ADDRESS TEST MODE
Enter <ruleset> <address>
> 0 yourid@yourhost
rewrite: ruleset  3   input: "yourid" "@" "yourhost"
rewrite: ruleset  8   input: "yourid" "@" "yourhost"
```

```
rewrite: ruleset  8 returns: "yourid" "@" "yourhost"
rewrite: ruleset  3 returns: "yourid" "<" "@" "yourhost" ">"
rewrite: ruleset  0   input: "yourid" "<" "@" "yourhost" ">"
rewrite: ruleset  3   input: "yourid" "@" "yourhost"
rewrite: ruleset  8   input: "yourid" "@" "yourhost"
rewrite: ruleset  8 returns: "yourid" "@" "yourhost"
rewrite: ruleset  3 returns: "yourid" "<" "@" "yourhost" ">"
rewrite: ruleset  0 returns: "^V" "tcp" "^W" "yourhost" "^X" "yourid" "@" "yourhost"
> ^D
prompt
```

The output shows that `sendmail` attempts to establish a TCP / IP connection with host `yourhost` (that is, the name following `"^W"`). This appears to be correct because `yourhost` is known to the name server. At this point, all you know is that `sendmail` is using the alias `yourhost` when it queries the name server, rather than using the official host name (`yourhost.domain.name`) that is displayed in the output from `nslookup`.

Next, look at the `sendmail.cf` file, and find the rule that produces the results returned by ruleset 0, as follows:

```
R$*<@$+>$*             $#tcp $@$2 $:$1@$2
```

The results returned by ruleset 0 are indicated on the right side of this rule. For example, `tcp` indicates a TCP / IP connection, shown as `"^V" "tcp"` in the output; `$2` represents the token that indicates the destination host's name, shown as `"^W" "yourhost"` in the output; and `$1@$2` represents the token that indicates the recipient's email address, shown as `"^X" "yourid" "@" "yourhost"` in the output. The character strings `$1` and `$2` represent the tokens that are created by the left side of the rule.

To ensure that `sendmail` always uses the official host name, change the rule to the following:

```
R$*<@$+>$*             $#tcp $@$2.$D $:$1@$2.$D
```

After making this change to the configuration file, you can send email addressed to `yourid@yourhost`, and it now works properly.

Cannot reply to e-mail originating from the Cray

A user attempts to reply to email received from a user on the Cray system, and the reply is returned as undeliverable. This problem started to occur only recently.

Using this information, you can conclude that one of the following problems exists:

- The `sendmail` program that exists on the Cray system contains an error.
- The `sendmail` program is not configured properly on the Cray system.
- The official host name for the Cray system is not being used.
- The `sendmail` program on the remote system contains an error.
- The `sendmail` program is not configured properly on the remote system.
- The routine that performs the host mapping function (`/etc/hosts` or the name server, depending on which one your site is using) is not correct, which causes the output that is returned by `gethostbyname`(3) to be in error.

The problem must be either with the method the Cray system uses to build its return address or with the method the remote system uses to return the email. (However, communication might be denied because of UNICOS security label policies.)

To isolate the cause of this problem, look at the changes that have been made in the components on the Cray system. For this example, assume that the only factor that has changed is that you now use the name server rather than the `/etc/hosts` file. Currently, you do not know what has changed on the remote system, but after you eliminate the possibility of an error in the Cray components, you can begin to look at the remote system.

Because the `sendmail` program is not changed, and this problem is new, you can assume that the `sendmail` program that exists on the Cray system is not in error. The next step is to determine which address this `sendmail` program is using as its return address. Either the official host name for the Cray system is not set up correctly, or the configuration file incorrectly constructs the name it is given.

The following example is a `sendmail` debugging tool that produces the sequence of events that `sendmail` executes when it delivers email to the remote system:

```
prompt> /usr/lib/sendmail -d8.99 -v yourid@yourhost
To: yourid@yourhost
From: u3441
asdf
u3441@yourhost... Connecting to yourhost.domain.name (tcp)...
220 yourhost.domain.name Sendmail 3.2/CRI-3.12 ready at Thu, 24 May 90 04:54:16 CDT
>>> HELO myhost.domain.name.domain.name
250 yourhost.domain.name Hello myhost.domain.name.domain.name, pleased to meet you
```

```
>>> MAIL From:<u3441@myhost.domain.name.domain.name>
250 <u3441@myhost.domain.name.domain.name>... Sender ok
>>> RCPT To:<u3441@yourhost.domain.name>
250 <u3441@yourhost.domain.name>... Recipient ok
>>> DATA
354 Enter mail, end with "." on a line by itself
>>> .
250 Mail accepted
>>> QUIT
221 yourhost.domain.name delivering mail
u3441@yourhost... Sent
prompt>
```

The `-d` option produces debug messages; the `-v` option requests a verbose execution method.

Note: When using this method to execute `sendmail`, you must enter the `To:` and `From:` lines, and the message text you are sending.

In the preceding output, the Cray host (that is, `myhost`) is telling the remote system (that is, `yourhost`) that its host name is actually `myhost.domain.name.domain.name`. This cannot be correct, because the domain name should appear only once. To determine whether the problem is in the method used to configure `sendmail` (that is, in the `sendmail.cf` file), you must first determine the method `sendmail` uses to construct this return address.

Then you can either consult the `sendmail` documentation or contact an analyst who is familiar with `sendmail`. Either way, `sendmail` uses the `$j` macro to define the local host's name, and thus, its return address. So you must locate the line in the `sendmail.cf` file that defines how `$j` is to be constructed. Locate the following line in the `sendmail.cf` file:

```
Dj$w.$D
```

The values that are returned from the `$w` and `$D` macros are concatenated with a period (.) between them. The `$w` macro is the official host name of the local host, and the `$D` macro is the local domain name.

To determine the name that is being returned as the official host name for the Cray system, you can execute `nslookup`. The following is a sample session:

```
prompt> nslookup
Default Server:  myhost.domain.name
Address:  128.162.75.1
```

```
> myhost
Server:  myhost.domain.name
Address:  128.162.75.1

Name:    myhost.domain.name
Address:  128.162.75.1

> ^D
prompt>
```

The `$w` macro returns `myhostname.domain.name`, which causes the `sendmail` configuration file to add incorrectly the value of `$D` (that is, the domain name) to the official host name.

Therefore, to resolve this problem, change the construction of the local host name in the `sendmail.cf` file by changing the line `Dj$w.$D` to `Dj$w`.

2.5 Trace Facility

This section describes the trace facility, which retrieves trace information from the TCP/IP and NFS kernel codes, and stores trace information in a file. This information can be formatted and used for program analysis and debugging.

A `socket`(2) system call that has the address family `AF_TRACE` creates a trace socket. A `connect`(2) system call, with appropriate options, connects this trace socket to the entries that are to be traced. The following types of parameters are accepted:

Parameter types	Description
Generic	Connects to a generic protocol or system layer, and all trace entries from that layer are sent to the user. The type field in the `sockaddr_trace` structure specifies `generic`; the mask field determines the layer. The following layers are supported: `if, ip, rawip, icmp, tcp, udp, nfs, idmap`
TCP specific	Connects to a specific TCP connection (defined by foreign and local addresses and ports). The `sockaddr_trace` type field specifies `tcp`; the `str_in` structure determines a connection.
UDP specific	Connects to a specific UDP connection. The options are similar to TCP specific tracing.

Interface specific	Connects to a specific interface. The interface name is provided in the `if_str` field of the `sockaddr_trace` structure.

Because they might be needed to trace multiple entries at one time, multiple connects are allowed on the trace socket. It is also possible for more than one user to trace one entry.

After the socket is created and connections are specified, three mbufs are queued up on the socket. Then the entities to be traced begin posting trace entries into these mbufs. When the mbuf is full, it is queued onto the socket-received space. The user can read the data by issuing a `read`(2) system call on the socket. The `close`(2) system call frees up the socket, the mbufs, and all structures related to the socket.

Each trace entry is prepended by a header that contains information that is used for formatting and data verification. The header includes the following fields:

Field	Description
`magic`	Magic cookie to use for synchronization if data is lost
`str`	Eight-character ASCII string, or an integer that the formatter can use
`id`	Unique ID for each trace entry
`type`	Type of entry (for example, `if`, `ip`, and so on)
`len`	Length of trace entry data
`subtype`	Used by the formatter
`rtc`	Real-time clock value of when the trace entry is made

For security reasons, only `root` is allowed to obtain trace information. Furthermore, because of the specialized nature of this socket type, only the following socket system calls are valid:

- `socket`
- `connect`
- `read`
- `close`
- `setsockopt`

- `getsockopt`

2.5.1 Collecting Trace Information

The `trcollect`(8) command collects TCP/IP and NFS trace information from the kernel. The syntax is as follows:

`trcollect` [*general options*] [*tcp/udp/interface options*] [*generic options*]

The following general options are accepted:

`-f` *tracefile*	Specifies a trace file in which trace output is stored. Redirecting standard output to a file has the same effect.
`-b` *buffersize*	Specifies the buffer (mbuf) size for the trace socket. Three mbufs per socket are queued. Note that the socket receive space must be greater than three times the buffer size. The default buffer size is 1 Kbyte.
`-r` *recvspace*	Specifies the socket receive space. The default socket receive size is 32 KB.
`-s` *timeout*	Specifies a time-out on read processes. By default, if no read occurs during the interval, after every 5 seconds the program flushes the current mbuf from the socket queue onto the receive space.

The following TCP/IP and UDP options are accepted:

`-t` *connectioninfo*	Specifies a TCP connection to be traced. The following options are accepted for *connectioninfo*:

`-fa` *faddress*	Specifies foreign address.
`-la` *laddress*	Specifies local address.
`-fp` *fport*	Specifies foreign port.
`-lp` *lport*	Specifies local port.

Note: Any options that are not specified are wildcarded. At least one option is required to trace TCP/IP connections.

`-u` *connectioninfo*	Specifies a UDP connection. The *connectioninfo* options are the same as those for the `-t` option.
`-i` *interface*	Specifies an interface to be traced (for example, `-i np0` traces all entries that go through the `np0` interface).

The following generic options are accepted:

`if`	Traces information on all interfaces.
`ip`	Traces information in the IP layer.
`tcp`	Traces all TCP / IP sockets in which the `SO_DEBUG` flag has been set.
`udp`	Traces all UDP sockets in which the `SO_DEBUG` flag has been set.
`icmp`	Traces information through the ICMP layer.
`rawip`	Traces information through the `rawip` layer.
`nfs`	Traces information through NFS.
`idmap`	Traces information through the NFS ID mapping scheme.

2.5.2 Formatting Trace Information

The `trformat`(8) program formats the trace information that is collected through the `trcollect`(8) command. The syntax is as follows:

`trformat [`*options*`] [`*types*`]`

The following options are accepted:

`-h`	Displays only the trace entry header information. The data part of the trace entries is skipped.
`-c`	Checks the sequences in the entries to ensure that entries are not missing.
`-f` *filename*	Specifies the trace file that is to be formatted. Redirecting from standard input has the same effect.
`-v`	Specifies that detailed information is to be printed.
`-x`	Specifies that trace entry data is to be printed in hexadecimal format.

`-n`	Specifies that addresses are to be displayed in dot format, and that ports are to be displayed as integers.
`-m hex_bitmap`	Specifies the types of ID map entries to be formatted.
`-s hex_bitmap`	Specifies the types of NFS entries to be formatted.

The *type* option lets the user select entries of the specified type. The options for this field are the same as for the *generic options* field of the `trcollect` command.

2.5.3 Obtaining Trace Socket Status

You can use the `netstat`(1B) command to view the status of a trace socket. `netstat -f trace` displays all active trace sockets. Following is an example of output from this command:

```
Active trace connections
Trace.ID Recv-Q State Msize Que    Rds/Flsh  Tracing
     133      0  CONN  1024   3    11/0      ip(generic)
```

Following is a description of the fields:

Field	Description
`Trace.ID`	Current trace ID.
`Recv-Q`	Amount of data (in bytes) in the socket receive queue.
`State`	Connection state. `CONN` indicates that a socket is connected to a tracing entity.
`Msize`	Size of the mbufs queued on the trace socket.
`Que`	Mbuf queue length.
`Rds`	Number of reads (not including flushes) on the socket.
`Flsh`	Number of times buffers were flushed from the socket queue to the receive buffer space.
`Tracing`	Shows data that is currently being traced by this socket.

2.6 Security Administration Basics

This section presents information about security administration that you must know to administer the Cray system on an ML-Safe network. The following aspects are described:

- Network security functional overview
- Identification and authentication
- Network security configuration
- Error messages

It is assumed that the reader is familiar with UNICOS security and the Cray ML-Safe configuration of the UNICOS system. For details of security administration, see *General UNICOS System Administration*.

2.6.1 Network Security Functional Overview

To provide secure communications, the Cray ML-Safe configuration of the UNICOS system and UNICOS security require hosts and networks to be defined in a network access list (NAL), thereby providing the security administrator or a system administrator a means to control networked accesses. The NAL describes the security labeling values associated with each remote host. This information is used to enforce the mandatory access policy on a network address basis.

UNICOS ML-Safe network operation is based upon protection at two levels: the interface to the network and the application interface.

Two mechanisms work at the network-interface level. These are the NAL and network interface label ranges. The NAL provides the system with detailed information about the capabilities of all remote hosts and networks with which the UNICOS system operates. The network-interface-label ranges ensure that communication is carried out only over network media that provide the correct level of assurance in accordance with the sensitivity of the data being transferred.

Each incoming or outgoing Internet Protocol (IP) datagram has a security label associated with it. This label can be implicit (as defined in the NAL) or explicit (as transmitted in IP security options). The datagram label is used to enforce the restrictions imposed by the NAL and the network-interface-label range and to ensure that data is delivered only to applications with the proper active label.

For incoming and outgoing datagrams, the datagram label must fall within the label range of the network interface that is used and within the label range specified in the NAL entry associated with the remote host.

At the application level, the UNICOS ML-Safe network services provide protection by requiring positive identification and authentication for all network transactions (except those that provide public information). In addition, the workstation access list (WAL) optionally controls application access based on user ID and/or group ID and the host from which access is desired.

Incoming and outgoing security violations and integrity errors are recorded in the security log, and Internet control message protocol (ICMP) responses are issued in accordance with the security rules defined in the Request For Comments (RFCs) for the particular IP security option.

The `spnet`(8) command manipulates the NAL, WAL, and Internet Protocol Security Options (IPSO) mapping tables in kernel memory. A properly authorized user may add, delete, and list entries in all three tables. The `/etc/config/spnet.conf` file is the repository for the contents of these tables. The `spnet.conf` file is a free-form text file, which the site administrator can maintain by using the installation tool or by editing directly.

2.6.1.1 Network Access List (NAL)

The network access list (NAL) contains information about remote hosts (or networks) that are authorized to connect to the local UNICOS system to perform sensitive processing or to access sensitive information. The NAL resides in kernel memory and is maintained through the use of the `spnet` command.

Each NAL entry can apply to either a single host or an entire network. The structure of the NAL is similar to the structure of the routing tables in the kernel. When a search for a host is performed, an exact match is tried first. If no exact match is found, it searches the network portion of the address. Finally, if no network entry is found, it uses the optional default NAL entry.

Each entry in the NAL consists of the following information about the remote host:

- Minimum security label
- Maximum security label
- Send and receive message authorization modes
- Security class
- Security option support (IP basic security option (BSO), Common IP Security Option (CIPSO), or none)
  - Domain of interpretation for CIPSO

- Protection authority (in and out) for IP BSO

The `spnet`(8) command creates and maintains the NAL, which is stored in the `/etc/config/spnet.conf` file.

If you are using the UNICOS ICMS, you can set up the NAL by using the `Configure system -> Multilevel security (MLS) configuration -> Network security options -> MLS Network access list (NAL) sets` menu.

2.6.1.2 IPSO Mapping Entries

With UNICOS security or the Cray ML-Safe configuration of the UNICOS system, using connections that support Internet Protocol Security Options (IPSO), every incoming and outgoing packet has its security label information translated through a designated IPSO map. The map is designated in the NAL entry for the remote host on the network to which the UNICOS security or the Cray ML-Safe configuration of the UNICOS system is communicating.

To conserve space in the IP protocol, security levels and compartments are represented by numbers rather than by their ASCII equivalent. IPSO maps are translation tables that allow the security administrator to control the numeric network representations and to adjust for differences in the implementation of labels among hosts on the network. Every incoming and outgoing packet from a UNICOS system to a particular host on the network has its label information translated through a designated IPSO map.

IPSO maps are numbered and, optionally, named. The map number, known as the Domain of Interpretation (DOI) number, is a 32-bit quantity that is included in every IP packet. A DOI is a collection of hosts that share a common definition of label values and meanings. The UNICOS system validates the DOI number against the DOI value in the NAL entry for the remote host. When the IPSO attribute in the NAL is set to basic, the DOI number is 0 implicitly. You can use the map domain name as a convenience within an `spnet` input file for associating NAL entries with maps. The UNICOS ICMS requires this name, but a default name is generated automatically when the `spnet` input file is imported, if it is missing.

When you are creating an IPSO map, you must consider the following issues:

- The host-internal numeric level values range as follows: `syslow`, 0 to 16, `syshigh`.
- The host-internal numeric compartment values are each a power of 2 ranging from 2**0 to 2**62.

- The numeric network representations for levels range from 0 to 255, and for compartments, from 0 to 65534.
- Each local level or compartment value must correspond to only one network value.
- Each network level or compartment value must correspond to only one local value.
- Network administrators must ensure that network-wide mappings are nonpermuting; that is, a translation from host A to host B to host C to host A would result in the original label.
- DOI 0 supports four levels and no compartments.

2.6.1.3 IP Security Options

An IP datagram always has an associated security label that is either implicit in the NAL definition or is explicitly carried with the packet that uses an IP security option in the options portion of the IP header. UNICOS security supports two explicit labeling types: the IP basic security option (IP BSO) (defined in Internet RFC 1108), and the Common IP Security Option (CIPSO) (defined in draft 2.2 of the Trusted Systems Interoperability Group (TSIG) document, *The Common IP Security Option*).

Both of these options represent security labels in a network representation that differs from the UNICOS internal representation. UNICOS security allows mapping table definitions that are used to translate between the network and host representations, and can adjust for differences in representations between hosts.

2.6.1.3.1 IP Basic Security Option (IP BSO)

The IP BSO is limited to providing the following sensitivity levels and no compartments:

- Unclassified
- Confidential
- Secret
- Top secret

By default, these levels map to the UNICOS levels 0, 1, 2 and 3, respectively. A site can override this default mapping, if required (see Section 2.6.3.6.5, page 228).

The IP BSO also carries a bit mask that represents a set of protection authorities. Network interfaces are marked with the valid protection authorities for which data can be sent or received on that interface. The NAL entries for hosts that use the IP BSO specify the protection authorities required on packets received from each host and the protection authorities that will be placed on datagrams destined for each host (see Section 2.6.3.6.3, page 223).

2.6.1.3.2 Common IP Security Option (CIPSO)

The CIPSO allows the representation of one of 256 levels and several of 65,535 compartments on each datagram. A Domain of Interpretation (DOI) identifies a mapping from a defining authority's level and compartment names to the network numerical representation. A DOI is identified by a 32-bit number, thus allowing for 232 possible DOIs.

No default mapping exists between UNICOS labels and CIPSO labels. To use the CIPSO, you must define an IP mapping table for each DOI with which communication is to occur, and load it into the kernel.

Each host that uses the CIPSO is identified with one particular DOI, which is specified in the NAL (see Section 2.6.3.6.3, page 223).

2.6.1.4 Workstation Access List (WAL)

The workstation access list (WAL) provides access control authorization for application-layer services. An entry in the WAL can apply to hosts or to an entire network. You can also create an optional default entry.

A WAL entry consists of a list of user / group pairs and the services for which those users and groups are allowed access. The user and / or the group can be wildcarded. The login user ID and primary group of the subject are used to perform WAL permission checks. First, a check is performed for the specific user and group, or for the user and a wildcard group. If no match occurs, a check is performed for the group. Finally, if no match has occurred, the *.* entry is used.

2.6.2 Identification and Authentication

Every subject (user) must be authenticated before being allowed access to UNICOS system services and sensitive data. For remote access, most subject authentications are initiated by `telnetd`(8) or `rlogind`(8), which connect to the `login`(1) process or by `ftpd`(8), which performs user validation. The NFS daemon conducts the authentication of subjects for NFS services. The NQS daemon conducts the authentication of subjects for NQS services.

2.6.2.1 Login Authentication

Each login request is based on a valid password entry (or SecurID value). Security clearances assigned are determined by the most restrictive set that is specified by the user database (UDB) in the `/etc/udb` file, the NAL, and the network interface over which the connection is made. The subject clearances that are granted for an authenticated user include the following:

- Active security label
- Minimum and maximum security label

When running one of the IPSOs, and when running the Cray ML-Safe configuration of the UNICOS system, the active security label of a session cannot be changed.

2.6.3 Network Security Configuration

This section describes the configuration operations for the UNICOS UNICOS network security. All of the information in Section 2.2, page 22, is applicable for a successful TCP/IP with UNICOS security enabled, and it should be read and followed prior to the UNICOS security configuration.

2.6.3.1 UNICOS Security Configuration Guidelines

The security configuration guidelines for the UNICOS security network software recommend that operations be performed in the following order:

1. Install the UNICOS operating system as released.
2. Install the network software as released.
3. Verify that the system operates correctly.
4. Verify the UNICOS security was installed with the default option settings.
5. Change the default options set for the UNICOS release, as necessary.

Note: The `named`(8) name server is not allowed under the Cray ML-Safe configuration of the UNICOS system.

When `named` is used with UNICOS security, there are special considerations. When the UNICOS system is put into multiuser mode, the `spnet`(8) command defines the NAL entries that allow socket communications. It reads the text version of the NAL from the `/etc/config/spnet.conf` file and translates the host names to host addresses. The `named` service will not be operating at this time because `localhost` sockets have not yet been allowed by a NAL entry. This means that name-to-address translations will be done from the `/etc/hosts` file (or `/etc/hosts.bin`, if it exists), which must be current and contain all of the hosts named in the `spnet.conf` file. The same is true for networks named in the `spnet.conf` file and translations from the `/etc/networks` files.

2.6.3.2 Network Security Options

The UNICOS security software for networks is activated when the security feature is enabled for the UNICOS operating system. This setting is maintained in low memory. See Section 2.6.3.6, page 222, for installation details.

Your site can alter the network security options described in the following sections to modify certain aspects of the operation of the ML–Safe network software. You can change these options by using the installation tool or by editing the kernel source file `config.h`. If you change any of these options, you must rebuild the UNICOS kernel.

Each configuration option is represented by a bit in the `SECURE_NET_OPTIONS` macro in the `config.h` file. To turn on an option, the option bit is OR'ed into `SECURE_NET_OPTIONS`.

Warning: When you are running the Cray ML-Safe configuration of the UNICOS system, some of the options described in this section require specific values. These options are noted where appropriate.

2.6.3.2.1 Strict B1 Networking: `NETW_STRICT_B1`

The `NETW_STRICT_B1` kernel option, when enabled, requires that all IP datagrams be unambiguously labeled. This means that hosts that do not use IP security options cannot have differing minimum and maximum labels in the NAL. If the system is configured as the Cray ML-Safe configuration of the UNICOS system, this option is turned on by default and must remain on in order to meet the requirements of the evaluation.

Note: When you are running the Cray ML-Safe configuration of the UNICOS system, the `NETW_STRICT_B1` configuration option must be turned on.

2.6.3.2.2 Multilevel Socket Compatibility: `NETW_SOCK_COMPAT`

The `NETW_SOCK_COMPAT` option, when enabled, automatically makes a socket created by an ML-Safe process into a multilevel socket. Set this option only if there are existing site-supplied ML–Safe processes that expect this socket behavior. Use of this option is strongly discouraged. If the system is running the

Cray ML-Safe configuration of the UNICOS system, this option is turned off by default and must remain off in order to meet the requirements of the evaluation.

An ML-Safe process has `PRIV_SOCKET` in a privilege assignment list (PAL) system or an ID of `root` in a `PRIV_SU` system. When the socket becomes connected, the socket's active label is frozen. When the socket is created, its minimum and maximum labels are set to the process minimum and maximum labels.

> **Note:** When you are running the Cray ML-Safe configuration of the UNICOS system, the `NETW_SOCK_COMPAT` configuration option must be turned off.

2.6.3.2.3 The `r` Command Compatibility: `NETW_RCMD_COMPAT`

The default behavior when UNICOS security is running is to require a host to be identified in both the `/etc/hosts.equiv` file and the user's `.rhosts` file, and the user must have the same ID on both systems. To operate with the traditional behavior, which requires a host to be identified in either the `/etc/hosts.equiv` file or the user's `.rhosts` file, set the `NETW_RCMD_COMPAT` option. This option can be set by using the installation tool or by editing the `cf/config.h` file.

2.6.3.3 NFS Configuration Options

Network file system (NFS) configuration options are maintained in low memory. The options are as follows:

Option	Description
NFS secure export	When enabled, this option indicates that `NFS` is permitted to export file systems that are labeled with security levels and / or compartments. This option is enabled with the UNICOS release. It is maintained in the `nfs_secure_export_ok` variable in the `config.h` file. A nonzero value indicates that exporting file systems that are labeled with security levels and / or compartments is permitted.
NFS remote read / write	Indicates that NFS is permitted to import remote file systems in read / write mode. This option is enabled with the UNICOS release. It is maintained in the `nfs_remote_rw_ok` variable in the `config.h` file. A value of 0 indicates read-only mode; a nonzero value indicates read / write mode.

UID mapping	See Section 3.1.6.8, page 264, and Section 3.1.6.9, page 266, respectively, for details of these options.

If you are using the UNICOS ICMS to select NFS export and import restrictions, use the `Configure system -> Multilevel security (MLS) configuration -> Network security options` menu with the `OK to export unclassified data via NFS` or `OK to import unclassified data via NFS` selections.

2.6.3.4 Restricting Access to Network Interfaces

UNICOS security supports a configuration option for restricting access to network interfaces to privileged processes. This option can be used to limit access to the OWS-E, MWS, and SWS workstations. To enable this option, invoke the `/etc/ifconfig` command with the `admin` keyword, as in the following example. To disable, use the `-admin` keyword.

```
# /etc/ifconfig np0 admin
# /etc/ifconfig np0
np0: flags=2041<UP,RUNNING,ADMIN> iftype vme
        inet 223.255.25.5 netmask ffffff00
level0-level16 compart 0-07777777777777777777
```

Note: When you are running the Cray ML-Safe configuration of the UNICOS system, the OWS-E interface must have the `admin` option set.

2.6.3.5 Labeling Network Interfaces

The network security installation process also includes setting the proper labels on network objects such as network interfaces. When the system is up and running, you can set these labels manually or you can use the UNICOS ICMS. For more information, refer to Section 2.2.9.6, page 109. To perform this task in multiuser mode, the administrator must be running as `root` in a `PRIV_SU` system, and / or have the appropriate security category active in a PAL system.

The network interfaces are labeled with minimum and maximum security labels.

Note: System daemons must be able to access the loopback interface at any label. Set the label range of the loopback interface the same as the system range. When the system is configured to enforce system high / system low labels, the label range must be `syslow` to `syshigh`.

2.6.3.5.1 Network Interface Label

Use the following command to assign a security label to a network interface:

`/etc/ifconfig` *ifname* `level` *min-max* `compart` *min-max*

Use the following command to display the security label of a network interface:

`/etc/ifconfig` *ifname*

In the following example the security label of network interface `np0` is displayed, and the label range of the loopback interface is set:

```
# /etc/ifconfig np0
np0: flags=41<UP,RUNNING>
        inet 201.201.201.2 netmask ffffff00
level level0-syshigh compart 0x0-0x9
# /etc/ifconfig lo0 level syslow-syshigh
        compart 0-07777777777777777777777
```

2.6.3.5.2 Network Security Configuration Example

The commands typically used when installing network security are as follows: `ifconfig`(8), `netstat`(1B), `spget`(1), and `spset`(1). This section contains a configuration example for a UNICOS security network (see Figure 16) and describes the installation commands that can be used. The example is UNICOS security with several hosts connected. Assume the following information:

- The UNICOS security host is a Cray Y-MP system with a serial number of 1007.
- Hosts connected to the `b-net` network support the BSO.
- Hosts connected to the `c-net` network support the CIPSO.

Note: It is always advisable to install and verify the released UNICOS system (with default option settings) before changing any default options.

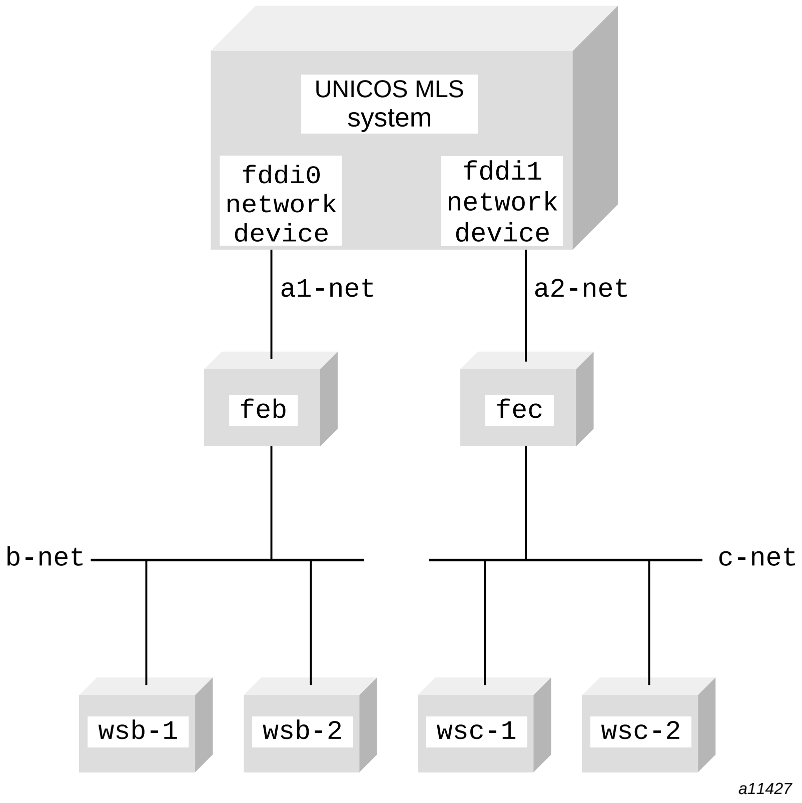


Figure 16. Configuration example

To initialize the network interfaces `np0` and `np1`, set up the `/etc/config/interface` file as follows:

```
np0 - inet crayb netmask 0xffffff00
np1 - inet crayc netmask 0xffffff00
```

To establish the routes shown in the example, use the following `route` commands:

```
/etc/route add b-net feb      *** b-net traffic thru feb
/etc/route add c-net fec      *** c-net traffic thru fec
```

To see how the network interfaces are configured, use the `netstat -i` command. Following is an example of a partial display:

```
# netstat -i
```

```
Name    Mtu     Network    Address

np0     32880   a1-net     crayb
np1     32880   a2-net     crayc
```

To obtain information and security labels for the network interface `np0`, use the `ifconfig np0` command.

To observe network route information, use the `netstat -rv` command.

2.6.3.6 Network Security Configuration Procedures

This section describes each of the steps you must follow for proper network security configuration. The steps are as follows:

1. Set options.
2. Check the system label.
3. Create the NAL.
4. Create the WAL.
5. Create the label map tables.
6. Label the network interfaces.

2.6.3.6.1 Step 1 - Setting Options

If you are going to use the UNICOS system without changing the installation options, you can skip the rest of this step and go directly to step 2.

When the UNICOS system has been installed as released and UNICOS security has been verified to operate correctly, you can make any changes to the released default configuration options (see Section 2.6.3.2, page 217).

You can use the UNICOS ICMS to set the configuration options through a menu presentation. After these options have been set, the UNICOS kernel must be rebuilt, shut down, and restarted.

If you are not using the UNICOS ICMS to set the installation options, you must set the options by making changes to the source in the `/usr/src/uts/cf.`*sn*`/config.h` file. After these options have been set in the source files, the UNICOS kernel must be rebuilt, shut down, and restarted. When the startup of the system is complete, proceed to the next step.

2.6.3.6.2 Step 2 - Checking the System Label

Because the system security label sets the security ranges for the UNICOS security environment, you should understand these security values before you label network interfaces and define the security ranges to be established for the remote hosts in the NAL. In some network configurations it might be necessary for ML–Safe daemons to communicate over the network (for example, when the Data Migration Facility (DMF) operates on one host and uses storage media on another host). In a case like this you must establish a range in the NAL and on the interface for the communication path to allow the `syshigh` label. To avoid runtime security violations, you should restrict the security label ranges applied to network objects to be within the system boundaries. However, a range that ends with `syshigh` includes all of the levels that are higher than the beginning of the range. Because of this effect, the use of `syshigh` in a label range should be minimized.

To see the current security label for UNICOS security, enter the `spget -s` command. This command produces the following sample output:

```
# spget -s

system minimum level is  0
                level0
system maximum level is 16
                level16
valid system compartments are 0777777777777777777777
                comp24
                comp39
                comp63
```

The names associated with the levels and compartments are configured in the site's `cf.` *nnnn* `/seclab.c` file. Remember the system's minimum and maximum security label when you label the network interfaces and when you set security labels for remote hosts in the NAL. Refer back to this step as necessary when you perform subsequent steps.

2.6.3.6.3 Step 3 - Creating the NAL

Before you create the network access list (NAL), identify the hosts and networks with which you want to communicate, and decide if you want a default entry (if you configured the `NETW_EXEMPT_NAL` flag in prior releases, you will probably want to do this). For each host or network identified, do the following:

- Determine the host classification. The host classification designates the capabilities and level of trust assigned to a remote host. The class value is one of the following: D, C1, C2, B1, B2, B3 and A1. By default, a host is classified as B1.

  Note: The class is used only as an indication of function, it does not necessarily indicate the evaluated rating of the host.

  Hosts classified as D are not trusted to perform any of the functions of mandatory access control (MAC), discretionary access control (DAC), auditing, or identification and authentication (I&A). However, a class D host must be administratively or physically protected such that it does not receive packets for another destination, and it must ensure that the correct source address is placed in all outgoing packets. A class D host can operate at only a single security label.

  Hosts classified as C1 satisfy the class D requirements and also provide I & A and DAC functionality. Despite the additional functionality, the UNICOS system treats class C1 and class D hosts the same, because C1 hosts do not perform auditing.

  Hosts classified as C2 satisfy the class D and class C1 requirements and also provide audit functionality. In addition, a C2 host must restrict TCP and UDP reserved port numbers (those less than 1024) to ML–Safe users with the required privilege. The UNICOS system allows the relegation of I&A to C2 hosts by allowing assertion of identity on these reserved ports. You might choose this class for workstations even when they do not provide auditing, so that they can provide I&A information to the UNICOS system. However, this classification is not allowed with the Cray ML-Safe configuration of the UNICOS system.

  Note: For r-series commands, such as `rlogin`, `rcp`, and `rsh`, the host must be listed as class C2 or higher to establish a connection. This is also true for remote line printing through `lpd`, and for NFS clients.

  Hosts classified as B1, B2, B3 or A1 satisfy all the D, C1, and C2 requirements and also enforce MAC policy. Thus, a host designated with B1, B2, B3, or A1 classification is allowed to have minimum and maximum labels that differ. If the `NETW_STRICT_B1` configuration is selected, such a host must use an IP security option.

- Determine whether IP security options are to be used. Either the BSO or the CIPSO can be used. In the case of the BSO, the allowed (incoming) and required (outgoing) protection authorities must be identified. For CIPSO, the DOI must be identified.

- Determine the appropriate label range. For multilabel systems (the minimum and maximum labels are not the same), the class must be B1 or higher, and an IP security option must be used (BSO or CIPSO). The security option is not required when the `‘Strict B1 evaluation rules’` option is not configured. This option is found in the `Configure system -> Multilevel security (MLS) configuration -> Network security options` menu in the installation tool.

  Note: The `localhost` NAL entry is an exception. It must have a label range that is the same as the system range and a class equal to B1, but it must not have an IP security option. When the system is configured to enforce system high and low security labels, the label range for local host must be `syslow` to `syshigh`.

- Determine the access modes. Specifying `send` allows TCP connections from the local system to the remote system to complete. Specifying `receive` allows TCP connections from the remote system to the local system to complete. If set to `none`, no communication is allowed with the remote host. If omitted, the mode attribute defaults to `send` and `receive`.

The NAL configuration data is located in the `/etc/config/spnet.conf` file. It is identified by the `nal` keyword, followed by the NAL data, enclosed in curly braces. The following code shows an example of the NAL portion of the `spnet.conf` file:

```
nal {
    ip host localhost  {
        class = B1;
        min label = syslow;
        max label = syshigh, 07777777777777777777777;
    }
    ip net default {
        min label = 0;
        max label = 0;
        class = D;
    }

    ip net basic-net { !default class = B1
        min label = 0;
        max label = top_secret;
        ipso = basic;
        auth-in = 0x8;!DOE
        auth-out = 0x8;!DOE
    }
    ip net cipso-net { !default class = B1
        min label = 0;
        max label = level16, software, accounting, engineering;
        ipso = cipso;
        doi = 1;
    }

    ip non-cipso-host { !default ipso = none
        min label = 0;
        max label = 0;
        class = C2;
    }
}
```

In the preceding example, two local networks use IP security options. One uses BSO and the other uses CIPSO. A default entry is defined, which allows any other hosts to access the UNICOS system at level 0 only. In addition, one host on the `cipso-net` network `non-cipso-host` does not use the `cipso` option; therefore, a host entry is defined with the minimum and maximum labels of 0.

The range of the local host entry must be equivalent to the system range. The range should be from `syslow` to `syshigh` on systems with the `SECURE_MAC` configuration parameter set.

NAL entries become active when they are loaded at start-up time, or they are loaded manually by using the `spnet add -nal` command.

If you are using the UNICOS ICMS, you can set up the NAL by using the `Configure system -> Multilevel security (MLS) definitions -> MLS Network security options -> Network Protocol Security Configuration -> MLS Network Authorization List (NAL) Sets` menu.

For more information on the NAL configuration and the operation of the `spnet` command, see the `spnet`(8) man page.

2.6.3.6.4 Step 4 - Creating the WAL

The workstation access list (WAL) controls application access authorization by host, user, and group identification.

The WAL configuration data is located in the `/etc/config/spnet.conf` file. It is identified by the keyword `wal`, followed by the WAL data, enclosed in curly braces. The following code shows an example of the WAL portion of the `spnet.conf` file:

```
wal {
    ip net default {
        *.* = all;
    }

    ip net network-a {
        0.* = none;
        fred.* = ftp;
        *.* = all;
    }

    ip host-b {
        *.* = none;
        sally.* = login, ftp;
        *.red = ftp;
        *.blue = login;
    }
}
```

In this example, a default entry, a network entry, and a host entry are defined. By default, all users in all groups are granted access to all WAL-controlled applications. On the network called `network-a`, user `root` is not allowed access, user `fred` is allowed only `ftp` access, and everyone else is allowed access to all WAL-controlled applications. For host `host-b`, user `sally` is allowed `login` and `ftp` access, users in group `red` are allowed `ftp` access, users in group `blue` are allowed `login` access, and everyone else is denied access.

WAL entries become active when they are loaded at start-up time, or when the `spnet add -wal` command is used to load them manually.

If you are using the UNICOS ICMS, you can set up the WAL by using the `Configure system -> Multilevel security (MLS) configuration -> MLS Network security Definitions -> Network Protocol Security Configuration-> MLS Workstation access list (WAL) Sets` menu.

For more information on the WAL configuration and the operation of the `spnet` command, see the `spnet`(8) man page.

2.6.3.6.5 Step 5 - Creating Translation Tables

This section describes creation of translation tables for hosts that support the IP basic security option (BSO) and for hosts that support CIPSO. The IP security option mapping tables are defined in the `/etc/config/spnet.conf` file. Multiple maps might be defined. They are identified by the keyword `map`, followed by map definitions, inside curly braces.

The following code shows an example of map definitions in the `/etc/config/spnet.conf` file:

```
map {
     basic {
        levels {
            0 = 0;
            1 = 1;
            2 = 2;
            3 = 3;
        }
    }

    cipso 1 {
        levels {
            public = 0;
            private = 1;
            proprietary = 2;
            syslow = 063;
            syshigh = 066;
        }
        compartments {
            accounting = 1;
            software = 2;
            engineering = 3;
        }
    }
}
```

In this example, two maps are defined. The first map defines the mapping for the basic security option. For the purpose of defining the map, network levels 0, 1, 2, and 3 correspond to the following basic levels, respectively: unclassified, confidential, secret, and top secret.

The `cipso` map definition defines a mapping for CIPSO domain of interpretation 1.

Translation tables become active when they are loaded at start-up time, or when the `spnet add -map` command is used to load them manually.

If you are using the UNICOS ICMS, you can set up the `cipso` map by using the `Configure system -> Multilevel security (MLS) configuration -> MLS Network security Definitions -> Network Protocol Security Configuration -> CIPSO Map Domain Sets` menu.

For more information on map configuration and the operation of the `spnet` command, see the `spnet`(8) man page.

2.6.3.6.6 Step 6 - Labeling the Network Interfaces

After UNICOS security is installed and operating, the security label of the network interfaces can be changed or set.

To set a security label on the network interfaces `lo0`, `np0`, and `np1` in the configuration example shown in Figure 16, page 221, enter the following commands:

```
/etc/ifconfig lo0 level syslow-syshigh compart 0-77777777777777777777
/etc/ifconfig np0 crayb netmask 0xffffff00 level 0-6 compart 0-7
/etc/ifconfig np1 crayc netmask 0xffffff00 level 0-syshigh compart 0-777
```

To verify the security label settings, display the security label of a network interface by entering the `ifconfig` *ifname* command, as follows:

```
# /etc/ifconfig np0

np0: flags=41<UP,RUNNING>
        inet 128.162.82.1 netmask ffffff00
level0-level6 compart 0x0-0x7
```

2.6.3.7 `inetd` Operation

The `inetd` operation has been modified so that it can be used safely in a UNICOS security environment. `inetd` opens each listen socket as a multilabel socket capable of receiving requests at any label that is valid for the user that is requesting a service. When a TCP connection request or UDP datagram is received, `inetd` forks the server at the same label as the incoming connection or datagram, and passes it as a single-label socket at that label.

On a system that uses privilege assignment lists (PALs), `inetd` leaves the `system` category active. ML–Safe servers should have a PAL that grants them the necessary privilege.

Warning: When you are running the Cray ML-Safe configuration of the UNICOS system, you cannot add ML–Safe servers to `/etc/inetd.conf` other than those supplied.

2.6.4 Error Messages

This section lists the error messages that are presented to the user when security violations are detected. The conditions that can produce the error messages are summarized, and the general corrective measures are stated.

These error messages can be presented for error conditions other than network security violations; therefore, this section also describes a method for searching for IP audit records in the security log. The administrator must analyze the security log entries to determine the actual security violation.

Finally, this section describes the steps you can use to isolate and correct the problems reported. These network security debugging guidelines should be applied in addition to those in Section 2.4, page 168.

2.6.4.1 Network Access Violations

The security violations detected for network access operations are always recorded in the security log and reported to the user with an error message. For the detected security violations, the administrator should further investigate the error messages reported to the user either by analyzing the security log or by checking for the conditions that are associated with the error message. To obtain the exact violation condition, the administrator must use the `reduce`(8) command to extract the actual network security violation. See Section 2.6.4.2, page 233, for more information on using this command.

Note: UNICOS NFS does not record all detected security violations in the security log.

The error messages presented to the user are as follows:

- `Connection timed out`
- `Host is unreachable`
- `Login incorrect`
- `Network is unreachable`
- `Permission denied`

- `Software caused connection abort`

This section describes the possible security violations that could generate each message.

2.6.4.1.1 `Connection timed out`

This message is usually displayed at the remote host when UNICOS security detects a security violation with the incoming request from the remote host. This message is presented when one of the following security violation conditions are sensed:

- The security label of the incoming datagram does not fall within the security label range of the UNICOS security host.
- An entry for the remote host does not exist in the NAL.
- The remote host specified in the NAL is not authorized to receive datagrams from this UNICOS system.
- The remote connection requires an IP security option and the incoming IP datagram does not contain the IP security option.
- A network-security-label to host-security-label translation error occurred.
- The remote host requires the CIPSO and a translation table is not available on UNICOS security for the remote host's domain of interpretation.

The `connection timed out` responses for the most common commands are as follows:

```
ftp sn7007
ftp: connect: Connection timed out

ping sn7007
sn7007 not responding

rlogin sn131
sn131: Connection timed out

telnet sn7007
Trying 128.162.82.1
telnet: Unable to connect to remote host:
Connection timed out
```

The `Connection timed out` message usually indicates a problem with the NAL entry for the remote host when it attempts to access the UNICOS security host. Your first isolation step should be to check the NAL values for the remote host. The security log for `netip` record type might also provide additional information.

2.6.4.1.2 `Host is unreachable`

This message can be displayed when routing by group ID fails.

2.6.4.1.3 `Login incorrect`

This message is issued by a user validation process (for example, `ftp` or `login`) when it detects that user access from the given remote host or to the requested service is not authorized in the WAL. Some of the network services that require authorization are `ftp` and `telnet`.

For a UNICOS system, the WAL can be used to allow or to disallow the use of the following services:

- `ftp`
- `login` (`telnet`, `rlogin`, and `rsh` without a command)
- `nfs`
- `nqs`
- `rexec`
- `rsh`

The other conditions that initiate the `Login incorrect` message are as follows:

- Login is denied because of a WAL violation
- Login process request to set the user's security values (through the `setusrv` system call) failed
- Maximum number of attempts to login were exceeded

2.6.4.1.4 `Network is unreachable`

This message can be displayed when the security label of incoming datagram does not match the network interface.

This message is usually issued when an IP datagram has a security label that does not fit within the security label range for the network interface. Your first isolation step should be to check the security labels of the network interfaces.

2.6.4.1.5 `Permission denied`

This error message is typically issued because of an error or security violation detected for an outgoing transmission request from a UNICOS security process; it is displayed when one of the following security violations is detected:

- The security label of the outgoing request does not fit within the security label range of the remote host in the NAL or on the interface.
- An entry for the remote host does not exist in the NAL.
- The remote host specified in the NAL is not authorized to receive datagrams from this UNICOS system.
- A security label mapping error was detected for an outgoing datagram.

The `Permission denied` message usually indicates a problem with the NAL entry for the remote host that the UNICOS security host process is attempting to access. Your first isolation step should be to check the NAL values for the remote host.

2.6.4.1.6 `Software caused connection abort`

This message is displayed when the security label of the incoming datagram is not within the security label range of the socket connection.

2.6.4.2 `reduce`(8) Command

The `reduce`(8) command provides detailed information on network security violations. For information about the security log, see the description of the UNICOS security feature in *General UNICOS System Administration*.

2.6.4.3 Problem Isolation Guidelines

This section describes solutions for problems that occur because of changes in the UNICOS security configuration or inconsistencies in the applied security privileges.

2.6.4.3.1 Session Hangs

When an interactive or batch session hangs, you receive no response to commands. This situation can occur when the NAL entries defined for your remote host were altered after your session started. If a NAL entry for your remote host changed to exclude your session's label from the entry's range, your UNICOS session no longer communicates with your remote host or workstation.

The session hang condition can also occur if network interfaces are altered such that the UNICOS process can no longer access the network.

2.6.4.3.2 Security Log Entries

The security log contains records of security violations detected by the kernel. Use the `reduce` command to recover network security violations recorded in the security log. For more security log information and examples, see the description of the UNICOS security feature in *General UNICOS System Administration*.

Network File System (NFS) [3]

The UNICOS network file system (UNICOS NFS) is a Cray software product that lets users share directories and files across a network of machines. Users of UNICOS NFS can use standard UNICOS I/O system calls, commands, and permission controls to access files from any file system. Similarly, other users of NFS can make use of UNICOS file systems from anywhere in the local network environment. UNICOS NFS can be used in diverse administrative environments through the use of the ID mapping facility. This facility is on by default in the UNICOS kernel. The user interface to UNICOS NFS is transparent.

UNICOS NFS supports version 2 and version 3 of the NFS protocol. However, in addition to the UNICOS license, a customer must purchase a separate contractual license for ONC+ to obtain a FLEXlm key. Without a FLEXlm key, a customer cannot access the NFS version 3 technology within UNICOS.

UNICOS NFS uses a server/client system to provide access to files on the network. A *server* is any machine that allows a part of its local disk space to be exported (made available for mounting on a host machine). A *client* is any machine that makes a request for an exported file system. When a user of UNICOS issues an I/O call (such as `read`(2), `write`(2), or `create`(2)) for a file that resides on a file system mounted by NFS, the call is transmitted to the server machine. When the server receives the request, it performs the indicated operation. For read or write requests, the indicated data is returned to the client or written to disk, respectively. This processing is transparent to users, and it appears that the file resides on a disk drive that is local to the UNICOS operating system.

UNICOS NFS supports Bulk Data Service (BDS), a nonstandard NFS extension. BDS improves large file transfer-speeds by providing direct I/O capabilities over the network for files that 100 MB or larger. For more information about BDS, see the `bds`(8) man page.

Both BDS Client and BDS Server are automatically shipped with UNICOS. A customer must contractually license BDS Client and/or BDS Server as separate items, each with their own separate license fee to obtain the FLEXlm keys to access BDS.

The following sections explain various aspects of UNICOS NFS:

- Section 3.1, page 236, provides system administrators with necessary information on activating and configuring NFS, setting up servers and clients, ID mapping, and security.

- Section 3.2, page 274, describes some common problems facing system administrators and suggests solutions.
- Section 3.3, page 287, describes the test suite that provides for early detection of UNICOS NFS problems.
- Section 3.4, page 293, describes factors that affect NFS performance, and methods for obtaining performance figures.

3.1 Administering UNICOS NFS

UNICOS NFS supports both NFS server and client capabilities. UNICOS NFS servers allow remote systems to mount local UNICOS file systems or directories; UNICOS NFS clients allow remote file systems or directories to be mounted locally. Users can then access and manipulate files in the usual way, subject to usual permission checks. The fact that parts of a file system might reside on various machines around the network is transparent to users. As system administrator, you control the use of these file systems. The following sections provide the information you need to activate, configure, and maintain NFS.

3.1.1 Activating NFS

If you are upgrading from UNICOS 9.0 and using the conversion utility, the NFS feature is on or off, depending on whether the feature was turned on or off in your UNICOS 9.0 configuration. Otherwise, the NFS feature is off by default.

If you are using the UNICOS Installation Configuration Menu System (ICMS) for your configuration, consult the `Configure System -> Major software configuration` menu for the menu item that turns on the NFS feature.

If you are not using the UNICOS ICMS for your configuration, you can turn on the NFS feature by modifying the `/etc/config/config.mh` file. Change the line that reads

```
#define CONFIG_NFS 0
```

to read

```
#define CONFIG_NFS 1
```

After you make this change, follow the rest of the system build procedures outlined in the *UNICOS System Configuration Using ICMS*.

3.1.2 Choosing a Configuration Method

The following sections describe three methods you can use to configure UNICOS NFS. Details of NFS server and client configuration are described in Section 3.1.3, page 238, and Section 3.1.4, page 241, respectively. Details of ID map configuration are described in Section 3.1.6, page 246.

3.1.2.1 UNICOS ICMS Configuration Method

You can use the UNICOS ICMS to configure NFS servers and clients.

Note: The UNICOS ICMS does not support configuration of NFS ID mapping. Refer to Section 3.1.6, page 246, for details of ID mapping.

As you use the UNICOS ICMS, the scripts and files that the UNICOS system supplies are updated for you.

Following is a description of the submenus that you can use to configure an NFS server:

- `Configure system -> Network configuration -> NFS configuration`

  This submenu lets you configure your system as an NFS server and create an `/etc/exports` file.

- `Configure system -> System daemons configuration -> System daemons table`

  This submenu configures daemons that are required for an NFS server by updating the `/etc/config/daemons` file.

Following is a description of the submenus that you can use to configure an NFS client:

- `Configure system -> Network configuration -> NFS configuration`

  This submenu lets you configure your system as an NFS client and create automount maps.

- `Configure system -> File System (fstab) Configuration -> NFS file systems`

  This submenu lets you configure the `/etc/fstab` file with a list of static NFS mounts. The `/etc/mountnfs` file that is called within the `/etc/nfsstart` script can mount these NFS file systems or directories at system startup.

- `Configure system -> System daemons configuration`

  This submenu lets you configure daemons required for an NFS client by updating the `/etc/config/daemons` file.

The help menus provide further assistance for using the UNICOS ICMS to configure NFS.

3.1.2.2 Manual Configuration Method

You can manually configure NFS by using the scripts and files that are supplied with the UNICOS operating system. The `/etc/nfsstart` script, which is called from the `/etc/netstart` script, is the script that allows manual configuration. After you have activated UNICOS NFS (see Section 3.1.1, page 236, for details of activating NFS), the `/etc/nfsstart` script performs the following actions:

1. Executes the `/etc/uidmaps/Set.domains` script, which either enables or disables ID mapping. See Section 3.1.6, page 246, for details on creating this and other ID mapping scripts.
2. Calls the `/etc/sdaemon` script to start the necessary NFS daemons in the `/etc/config/daemons` file. You must manually update this file. All input required for this file is described in Section 3.1.3, page 238, and Section 3.1.4, page 241.
3. Mounts selected remote NFS file systems or directories by calling the `/etc/mountnfs` script. You must manually update this script. All input required for this script is described in Section 3.1.4, page 241.

3.1.2.3 Local Script and File Configuration

You can configure UNICOS NFS by using local scripts and files. Details of this method of configuration are given in Section 3.1.3, page 238, Section 3.1.4, page 241, and Section 3.1.6, page 246.

3.1.3 Setting up a UNICOS NFS Server

A UNICOS NFS server is a machine that can export its own file systems and directories to another machine (an NFS client). Following are the steps required to configure your system as an NFS server:

1. Export file systems and directories

As super user, enter the mount-point path name of the file systems and directory hierarchies that you want to export in the `/etc/exports` file. (See `exports`(5) for the file format details.)

For example, to export `/usr/src/mybin` to `machine7` and `machine9`, and to export `/usr/man` to all machines, add the following lines to the `/etc/exports` file (or use the UNICOS ICMS):

```
/usr/src/mybin  -access=machine7:machine9
/usr/man
```

As shown, if no machines are specified for a file system, the file system is exported globally (that is, any machine can mount it).

The `exportfs`(8) command activates export entries in the `/etc/exports` files. By default, the `exportfs` command reads the `/etc/exports` file and puts an entry for each valid export in the `/etc/xtab` file. The `mountd`(8) command reads the `/etc/xtab` file to determine access rights.

The `exportfs` command is usually run during system startup from an entry within the NFS group of the `/etc/config/daemons` file. However, if a change is made to the `/etc/exports` file while the system is running, the `exportfs` command must be executed to make the changes effective. For example, the following command activates or changes a single export in the `/etc/exports` file:

`exportfs` *pathname*

The *pathname* variable specifies the full path name of the file system or directory to be exported. Any options associated with this export are read from the `/etc/exports` file.

The `/etc/exports` file must have unique entries for each file system; if entries are repeated, only the last entry that is read is valid. For example, consider the following commands:

```
/tmp orion, stardust
/tmp starship
```

In this example, only `starship` has access to `/tmp`.

2. Set up NFS server daemons

   The following daemons are required for an NFS server (this list assumes that the `/etc/portmap` daemon for RPC was already started through the TCP group in the `/etc/config/daemons` file):

Daemon	Description
`mountd`	The NFS mount daemon handles incoming NFS mount requests from NFS clients. When the NFS mount request is received, `mountd` reads the `/etc/xtab` file to determine which file systems and directories are available to export and to determine the NFS client systems to which these files can be exported (see the preceding step regarding `/etc/xtab`). The `mountd` daemon is usually run during system startup from an entry in the NFS group of the `/etc/config/daemons` file.
`nfsd`	The NFS daemons handle NFS client file access requests after an NFS file system is mounted successfully. Typically, four `nfsd` daemons are run; these daemons are usually started during system startup from an entry within the NFS group of the `/etc/config/daemons` file, and it has no options. Following is a sample command that shows a request for four processes: `/etc/nfsd 4`
`cnfsd`	The `cnfsd` (Cray `nfsd`) daemon starts NFS server daemons, which use a modified NFS protocol that allows protocol extensions and eliminates most eXternal Data Representation (XDR) processing. One advantage of using `cnfsd` is that it allows Cray systems to communicate with other Cray systems with 64-bit file offsets; `nfsd` uses the NFS protocol standard of 32-bit file offsets. `cnfsd` does not provide a significant performance enhancement over `nfsd`; `cnfsd` is intended as an NFS functionality enhancement between Cray systems. The `cnfsd` daemon is intended for use only between Cray systems. For additional NFS communication to systems that are not Cray systems, `nfsd` must also be started in conjunction with `cnfsd`.

`pcnfsd`	This daemon is required on an NFS server if any NFS clients are personal computers (PCs). Because PC users do not have user IDs, it is necessary to perform special user authentication when an NFS request comes from them. The `pcnfsd` daemon runs continuously on an NFS server system to service PC NFS requests for user authentication and print spooling.
`kerbd`	The `kerbd` daemon is required if `AUTH_KERB` NFS is configured. The daemon handles requests from the kernel NFS and sends requests to and from the Kerberos key distribution center (KDC). `kerbd` also maps Kerberos user names into local user and group IDs.

Note: Remote Procedure Call (RPC) server applications (for example, `mountd` and `nfsd`), communicate with `portmapper` on the last local Internet interface that is up and running (usually `loopback`). With UNICOS security, the application must be allowed to connect with a socket on this interface. This is done by adding an explicit NAL entry in the `spnet.conf` file for the corresponding host name or a default entry. You can use the `netstat -iv` command to determine the IP address of the last up and running interface connection.

3.1.4 Setting up a UNICOS NFS Client

A UNICOS NFS client is a machine that can mount remote file systems and directories from another machine (an NFS server). To configure your system as an NFS client, the system must gain access to an exported file system and files must be set up to perform desired mounts during system startup. Automounter maps can also be set up for dynamic mounting. The following sections describe these procedures.

3.1.4.1 Mounting a Remote File System

To gain access to an exported file system, an NFS client simply mounts the file system as if it were located on a local disk. By using the `mount`(8) command, you can mount any exported file system or directory on your machine if you can reach its NFS server over the network and if your machine is included in the `/etc/exports` list for that file system, or the file system is exported globally.

Using the `automount`(8) command, you can cause specified file systems to be automatically and transparently mounted whenever a file or directory within that file system is opened (see Section 3.1.4.2, page 244, for more details on automatic mounting).

After a file system is mounted, it is accessible to users as if it were a local subdirectory.

To mount a file system or directory from an NFS server, become the super user or activate one of the administrative categories with UNICOS security, and type the `mount`(8) command with the desired options. For example, to mount the man pages from remote machine `elvis` on the local directory `/usr/man`, you can type the following:

```
mount -t NFS -o bg,soft,rsize=8192,wsize=8192,nreadah=2 \
    elvis:/usr/pubs/man /usr/man
```

The `bg` argument indicates that if the NFS mount fails, the NFS mount request should be tried repeatedly in the background. Without either the `bg` or the `soft` arguments, failed NFS mount requests are retried up to the maximum number of retries specified (or set by default) on the `mount` command; these retries occur in the foreground.

The `soft` argument indicates that if the server does not respond to either an NFS mount request or an NFS request to an already mounted file system, the requested operation fails with an error. This argument prevents processes on the client from hanging while waiting for the NFS server to respond.

It is strongly recommended that all NFS mounts from a Cray system be soft mounts because hard mounts, indicated by the `hard` argument of the `mount` command, continue to try either mounting the NFS file system or accessing that mounted system in the foreground until the NFS server responds to the NFS request. This can cause processes and especially system startups to hang until the NFS server responds. It is further recommended that you avoid mounting NFS file systems directly on the root directory or any other system-critical directory.

The `intr` argument can be used with the `hard` argument at mount time, which lets you interrupt a process that is hung while waiting for the NFS server to respond.

The `rsize` and `wsize` arguments set the read and write buffer sizes, respectively, to the specified number of bytes. The default value for both `rsize` and `wsize` is 8192 (8 Kbytes). This value is adequate for most NFS servers. However, when the server is also a Cray system running the UNICOS system, setting `rsize` and `wsize` to 32768 improves NFS performance. The `nreadah` argument sets the number of `rsize` asynchronous read-aheads. The default value of this argument is 1. See Section 3.4, page 293, for more details on `rsize` and `wsize`.

At system startup, you can mount frequently used file systems and directories by placing mount entries for them in the `/etc/fstab` file (see `fstab`(5)) and invoking the `/etc/mountnfs` script from within `/etc/nfsstart`. The `/etc/nfsstart` script is called from the `/etc/netstart` script. The `/etc/mountnfs` file could contain, for example, the following lines:

```
mount /usr2 &
mount /usr/man &
```

The corresponding entries for the NFS file systems in the `/etc/fstab` file might be as follows:

```
titan:/usr2     /usr2     NFS bg,soft,rw,rsize=8192,wsize=8192,nosuid
venus:/usr/man  /usr/man  NFS soft,nosuid
```

If no `fstab` entry exists, the `/etc/mountnfs` file could contain the following lines:

```
mount -t NFS -o bg,soft,rw,rsize=8192,wsize=8192,nosuid titan:/usr2 /usr2 &
mount -t NFS -o soft,nosuid venus:/usr/man /usr/man &
```

Note: Performing NFS mounts in the background (this is done by starting `/etc/nfsstart` in the background or by using the `&` shown in the preceding example) ensures that the remainder of system startup completes in a timely manner, even if the remote NFS server systems do not respond to the mount requests.

The `biod` client daemon handles asynchronous block I/O. The `biod` daemon attempts to collect contiguous data from system buffers and write them to the network in `wsize` length sections.

On Cray systems, the `biod`(8) daemon enables asynchronous write-behind and read-ahead processes that can significantly improve read and write performance. The `biod` daemon is usually run during system startup from an entry within the NFS group of the `/etc/config/daemons` file, and it has no options, except the number of `biod` daemons to start. For optimal performance, the number of `biod` daemons should never exceed the total number of static client handles. Following is a sample command that shows a request for four processes:

```
/etc/biod  4
```

3.1.4.2 Automount Facility

Note: The use of the automount facility is not supported with the Cray ML-Safe configuration of the UNICOS system.

The automount facility automatically and transparently mounts and unmounts an NFS file system as it becomes necessary. When a user on an NFS client machine running the automount facility enters a command that accesses a file or directory that belongs to a remote file system, the remote file system is automatically mounted. When the automatically mounted remote file system has not been accessed within a period of time, the file system is unmounted automatically.

The `automount`(8) command does not consult the `/etc/fstab` file for a list of remote file systems or directories to mount, but instead, has its own set of configuration files known as *maps*. Therefore, to enable the automount facility, you must first create map files. See `automount` for the format of these files and a description of the command.

The `automount` daemon is required if the NFS client will be running the automounter. The `automount` command is usually run during system startup from an entry within the NFS group of the `/etc/config/daemons` file. Following is a sample `automount` command:

```
/etc/automount -m -f /etc/auto/auto.master
```

This is a typical `automount` command because the UNICOS system requires the `-m` option and the use of an automount master file.

Note: Any automount options listed within the indirect map entries override all options listed in the master map for that entry. If you are using `automount` when running ID mapping on an NFS client, you must define `loopback` or `localhost` as an ID mapping domain.

3.1.4.3 Protocol between Cray Systems

When processing is between two Cray systems, you can use a modified NFS protocol (which the `cnfsd`(8) daemon uses) to reduce the CPU time required to process an NFS request. The removal of XDR processing makes this reduction possible. You can access much larger files across NFS with this protocol than with the standard NFS protocol because all file size and file offset fields within the modified protocol are a full 64 bits. Another advantage of using `cnfsd` is that it uses 32 Kbytes read and write sizes; `nfsd` defaults to 8 Kbytes. You can use the modified NFS protocol between Cray systems by specifying the `-o` option and `cray` operand with the `mount`(8) command on the Cray NFS client when mounting a Cray NFS server. Also, you must start at least one cnfsd process on the Cray NFS server. See mount(8) and cnfsd (see nfsd(8)) for more information.

3.1.5 Typical UNICOS NFS Layout

The following output from two mount(8) commands demonstrates the layout of UNICOS NFS in a typical environment, showing both the Cray client and a client that is not a Cray client. The first example is from a Cray system called cray2. It shows the local file systems and the NFS mounted file system /nfs/titan (exported from server titan). The output from the mount operation is followed by a listing of the mounted file system.

Example 1: cray2 as client, titan as server, as seen from the cray2 system:

```
cray2%  /etc/mount
/ on /dev/dsk/root  read/write on Mon Apr 4 06:55:07 1988
/usr on /dev/dsk/usr read/write on Mon Apr 4 06:55:35 1988
/u on /dev/dsk/u  read/write on Mon Apr 4 06:55:35 1988
/nfs/titan on titan:/usr/titan read/write,rsize=8192,wsize=8192 on Mon
   Apr 4  06:55:45 1988
cray2% ls -l /nfs/titan
total 39
drwxr-x---     22  btk          network 1536 Mar 31 10:16 btk
drwxr-xr-x      9  common       cray2    512 Feb  1 08:32 X
drwxr-xr-x     37  mer          netqa   2560 Mar 31 15:50 mer
drwxr-xr-x      5  prb          cray2   3072 Apr  1 22:51 prb
drwxr-xr-x     23  wtg          appl    1024 Mar 23 13:32 wtg
```

Example 2: titan as client, cray2 as server, as seen from the titan system:

```
titan% /etc/mount
/dev/xy0a on / type 4.2 (rw)
/dev/xy1c on /usr.MC68020/titan type 4.2 (rw)
cray2:/u on /usr/cray2/u type nfs (rw,soft,bg)
titan% ls -lg /usr/cray2/u
total 32
drwxr-x--x     26  btk          secure  1715 Jun 10  1984 btk
drwxrwxr-x     59  common       os      1089 Jan  7  1987 X
drwxr-xr-x     41  mer          netqa   3134 Apr 13 16:45 mer
drwxr-xr-x     12  pfh          starter  557 Jun  2  1986 pfh
```

```
drwxr-x---      2  slevy        msc      512 Feb 18  2:11 slevy
drwxr-xr-x     35  wtg          network  544 Apr  4 10:06 wtg
```

3.1.6 ID Mapping

UNICOS NFS includes an ID mapping facility that allows the use of NFS in diverse administrative environments. Traditional NFS environments make use of the Sun Microsystems network information service (NIS) distributed look-up service to provide for various network management functions. The user space that NIS provides is flat; that is, a given ID number always refers to the same user or group. This flat user space is necessary because NFS transmits user and group identifiers in binary form, and it provides no translation services for these values.

Cray systems, however, are often shared by many different administrative environments, making the creation of a single administrative space for user and group identification technically or organizationally difficult, if not impossible. A given ID number can refer to different users or groups in different administrative environments. To meet the needs of these environments, ID mapping was developed.

ID maps contain an equivalent remote ID for each local ID in a map. There are two types of ID maps: user ID maps and group ID maps. For every user ID map, a corresponding group ID map exists.

ID mapping domains associate Internet addresses with a particular pair of user and group ID maps. When an NFS request is sent to or received from an address within an ID mapping domain, the pair of ID maps associated with that ID mapping domain can be used to replace the IDs in the request.

ID mapping can also be used to control access to the local Cray system through NFS by allowing requests only from certain Internet addresses, or by restricting permissions for certain users at these addresses.

Figure 17 is a diagram of the ID mapping function as it relates to UNICOS NFS system interfaces.

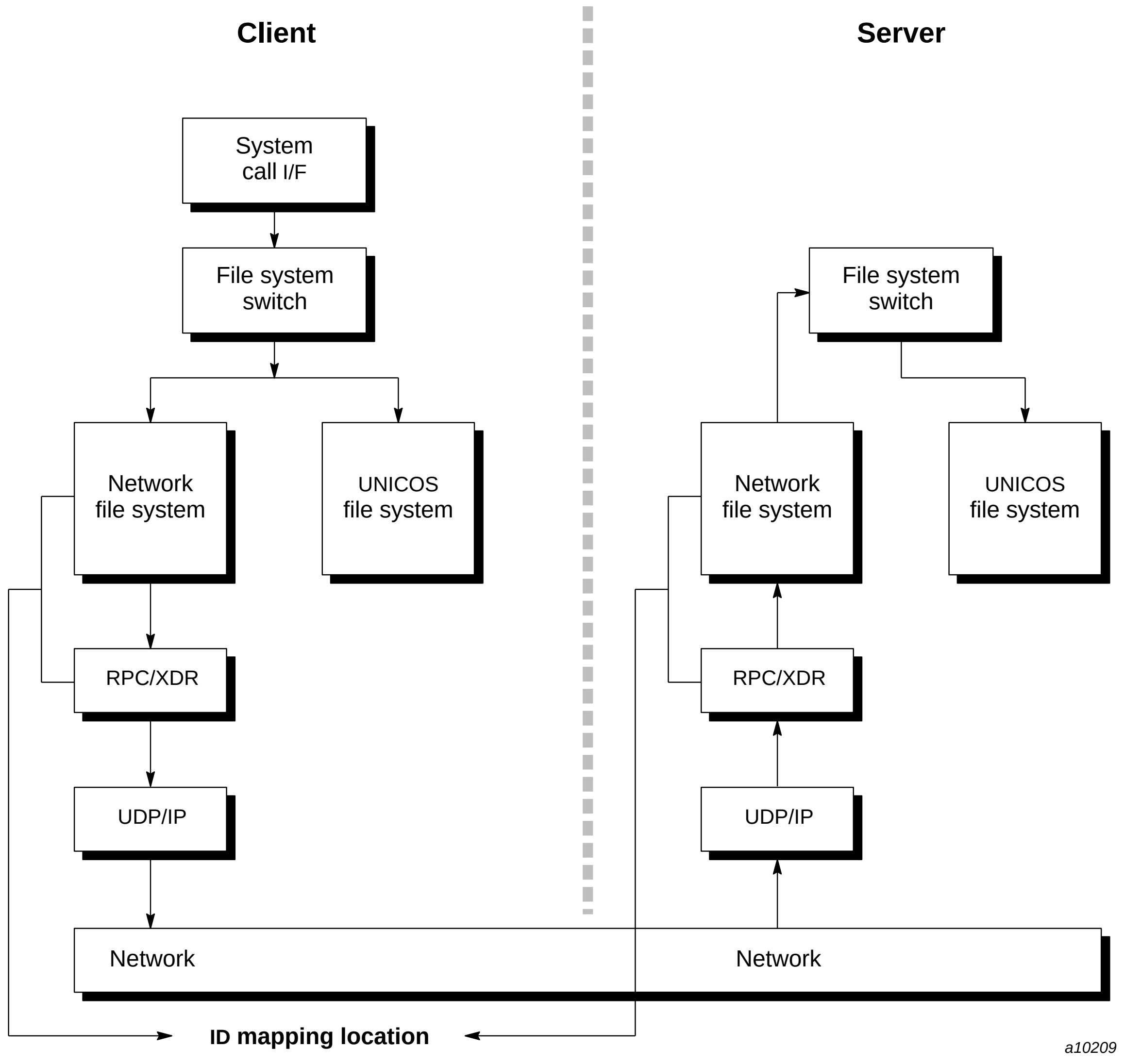


Figure 17. System interfaces and ID mapping

3.1.6.1 Disabling ID Mapping

ID mapping in the UNICOS kernel is on by default. The `nfsidmap`(8) command disables the use of ID mapping at run time. NFS ID mapping is not required when the Cray system and all other systems using NFS use the same user ID space.

To disable NFS ID mapping, create an executable `/etc/uidmaps/Set.domains` script that contains the following line:

```
/etc/uidmaps/nfsidmap -d
```

3.1.6.2 Configuring and Using ID Mapping

Configuring and using NFS ID mapping is a site-dependent function. However, you should always use the following basic steps to configure NFS ID mapping:

1. Obtain the `passwd`(5) and `group`(5) files from each remote administrative environment for which IDs will be mapped.
2. Use the `passwd` and `group` files from the local Cray system, along with those obtained from the remote systems, to create the user and group ID map files between the remote domains and the local Cray system.
3. Load the ID maps into the kernel, and define the ID mapping domains.

If a remote administrative environment is the same as that on the local Cray machine, the creation of an ID map for use between these machines (steps 1 and 2) is not necessary unless you are running with UNICOS security. With UNICOS security, ID maps are required for all remote environments.

The `/etc/uidmaps` directory exists to store ID mapping commands and associated files for ID mapping. In an environment without UNICOS security, the password and group files can be collected into this directory and processed. This directory also contains the ID mapping commands, the administrative shell scripts, and files that contain the constructed ID maps that will be loaded into the kernel. In a UNICOS security environment, the `/etc/uidmaps` directory is created with a `syshigh` label to protect the commands and maps after they are created. A directory with a `syshigh` label is unsuitable for remote copies from the network, so a separate directory must be created for collecting password and group files from remote systems.

There are two `/etc/uidmaps` subdirectories. The `/etc/uidmaps/users` directory should contain all `passwd` files from the remote administrative domains that are being mapped, and any exceptions files (explained in "Exceptions file," page Section 3.1.6.4.4, page 253). The `/etc/uidmaps/groups` directory should contain all `group` files from the remote administrative domains that are being mapped, and any exceptions files.

3.1.6.3 Network Description Example

The example used throughout this section shows how to configure and use the UNICOS NFS ID mapping facility. The procedures (shell scripts) shown can be used on all Cray machines in the example. Although the sequence of the commands shown is common to any Cray system using ID mapping, the specific contents of the procedures might vary according to site standards.

Assume the following network consisting of three Cray machines configured to perform various types of ID mapping, a single machine (not a Cray machine) with its own administrative domain, and several workstation networks using different NIS domains:

Name	Description
`groucho`	A Cray machine with its own administrative domain.
`chico`	A Cray machine with its own administrative domain.
`harpo`	A Cray machine with its own administrative domain.
`zeppo`	A different type of machine with its own administrative domain; that is, the assigning of login names and group names (and binary values associated with those names) occurs separately from those operations on other machines on the network. This machine is also running a version of UNIX, rather than another operating system.
`nfl`	A network of workstations that share a single NIS domain for both the `/etc/passwd` and `/etc/group` files. The name `nfl` was chosen because `nfl` is a file server that is the NIS master for the password and group NIS databases.
`disney`	A network of workstations similar to `nfl`, but with separate administration using NIS.

Each machine name is the host name of a representative machine for its administrative domain. It is also the name that is chosen for each ID mapping domain.

3.1.6.4 Setup, Creation, and Maintenance of ID Map Files Example

User and group ID map files are built from the passwd and group files of the local Cray system and of each remote administrative domain. Mappings are created for each user or group on the Cray system that matches a user or group on a remote administrative domain. Each mapping is placed in either a user ID map file or a group ID map file.

ID map files are built by using the nfsmerge(8) utility. The Get.domains and Merge.domains scripts control the creation of user maps. These scripts, which you must configure for local systems, correspond to steps 1 and 2 of configuring ID mapping domains, as described in Section 3.1.6.2, page 248. For these two steps, it is not necessary to run a kernel with ID mapping configured.

Note: You must manually supervise the running of Get.domains and Merge.domains to ensure that they actually work, because the scripts contain no error handling of their own. If they are run automatically and they encounter errors, the resulting user ID maps might not be valid or secure.

3.1.6.4.1 ID Map File Setup

The Get.domains script copies passwd and group files from the local system and from a representative of each remote administrative domain to the local system. You are free to use any utilities or mechanisms to create these files. Usually, rsh(1), rcp(1), or ftp(1B) is used to copy files from the remote systems. (If rsh(1) or rcp(1) is used, remote execution from the local Cray machine must be allowed on all remote machines.) If a remote machine has an operating system other than UNIX or one of its derivatives, the administrator of that machine must construct the equivalent passwd and group files to enable the nfsmerge(8) utility to create an ID map file.

The following is an example Get.domains script, which should be run whenever you are notified that users or groups were added to any of the remote domains:

Note: If your site uses ID mapping, root (uid equals 0) must be contained in the passwd(5) files to be used for the mapping.

The Get.domains script is historically stored at /etc/uidmaps/. This script is unique to each site's administration policies.

```
% cat /etc/uidmaps/Get.domains
```

```
#
# Script to collect and sort password and
# group files from the various domains referenced.
#
#    This for loop collects information from non-NIS hosts
#

USERS=/etc/uidmaps/users
GROUPS=/etc/uidmaps/groups

for sv in groucho chico harpo zeppo
do
   echo "Getting passwd from $sv"
   remsh $sv cat /etc/passwd | sort -t: +0 -1 -o $USERS/passwd.$sv
   echo "Getting group from $sv"
   remsh $sv cat /etc/group | sort -t: +0 -1 -o $GROUPS/group.$sv
done
#
#  This loop collects information from NIS hosts
#
for yp in nfl disney
do
   echo "Getting passwd from $yp"
   remsh $yp ypcat passwd | sort -t: +0 -1 -o $USERS/passwd.$yp
   echo "Getting group from $yp"
   remsh $yp ypcat group | sort -t: +0 -1 -o $GROUPS/group.$yp
done
```

The created files are called `passwd.` *domain* and `group.` *domain*. (As with the copying of files, this procedure requires that each remote machine allow remote execution from the local Cray machine.) Each of these files is sorted on the user or group name. It is not necessary to sort the `passwd` or `group` files before making the ID maps; however, sorting generally speeds the creation of the maps.

3.1.6.4.2 ID Map File Creation

The `Merge.domains` script creates the ID map files. This script calls the `nfsmerge`(8) utility to create the ID map files between the local Cray machine and the remote administrative domains. The `nfsmerge`(8) utility uses the `passwd` and `group` files (usually a copy of each) from the local Cray machine and from a remote administrative domain to create a mapping between the numerical user and group ID values on the two domains, using login and group names for comparison. It expects, for example, that the login name `grumpy` on the local Cray machine and on the remote administrative domain refers to the same user. The same is true for groups.

3.1.6.4.3 ID Map File Maintenance

Rerun `Get.domains` periodically to update the map files (see Section 3.1.6.4.1, page 250). The following `Merge.domains` script should be run after any update of the local copies of the `passwd` and `group` files by the `Get.domains` script. It should also be run when an exceptions file changes (see Section 3.1.6.4.4, page 253).

> **Note:** With UNICOS security, NFS ID maps contain mandatory access control (MAC) configuration information. Because these maps contribute to enforcement of MAC policy, you must protect these maps and the scripts that produce them by labeling them with the `syslow` label. To safeguard against security risks, the maps must also be manually inspected each time they are created. This is because the maps are constructed from password and group files that might not be protected by the `syslow` label when they are collected from the other systems on the network. The method of inspection and the degree to which that process can be automated is site-dependent.

To meet this labeling requirement, every script or program used in ID map generation must have a `syslow` label; they must be executed at `syslow`, and the resulting map files must have a `syslow` label. The NFS commands in the `/etc/uidmaps` directory and the directory itself are automatically installed with this label. These commands are privileged to access the `syslow`-labeled ID map files and to load the maps into the kernel. To use these commands, you must have the `secadm` category active. The `nfsmerge`(8) utility, which creates ID map files, labels them with its process-execution label. However, existing ID map files are overwritten without their label being changed. Therefore, you should remove all existing ID map files at the start of the ID map generation process.

The following is an example of a `Merge.domains` script, which is historically stored at `/etc/uidmaps`. This script is unique to each site's administration policies.

```
% cat /etc/uidmaps/Merge.domains
```

```
#
#  Script to create ID maps from the sorted passwd and group files
#  for each administrative domain.
#

HOST=‘hostname‘
```

```
USERS=/etc/uidmaps/users
GROUPS=/etc/uidmaps/groups
CMD=/etc/uidmaps/nfsmerge

for cray in groucho chico harpo
do

    for domain in groucho chico harpo zeppo nfl disney
    do
      if ‘test $cray != $domain‘      then
        echo "Creating user and group ID maps between $cray and $domain"
                                                    | tee l.$cray.$domain
        $CMD    -l $USERS/e.$cray.$domain -u u.$cray.$domain
                -e $GROUP/e.$cray.$domain -g g.$cray.$domain
                $USERS/passwd.$cray $USERS/passwd.$domain
                $GROUPS/group.$cray $GROUPS/group.$domain >> l.$cray.domain

    done
  fi
done
```

3.1.6.4.4 Exceptions File

Users are likely to have the same login name wherever possible, even though they might use several machines from different administrative environments. However, if a user has a login name in the remote administrative environment that differs from that on the local Cray machine, that user ID can be mapped into an exceptions file. The same applies to group ID mapping.

The exceptions file should contain a list of name pairs, one pair per line, the names separated by white space. The name pairs are of the following form:

local_name   *equivalent_remote_name*

For example, user Big Bad Wolf has the login name `bbw` on machine `groucho` and the login name `wolf` in the `disney` NIS domain. The exceptions file on `groucho` could be called `/etc/uidmaps/users/e.groucho.disney` and could contain the following entry:

```
bbw  wolf
```

Use of this exceptions file when making the map file between `groucho` and the `disney` NIS domain would ensure that the user ID mapping for Big Bad Wolf is placed in the map file.

If *equivalent_remote_name* is not specified in the exceptions file, it is considered the same as *local_name*. This feature is useful for restricting maps to use only the names in the exceptions files (see the `-E` or `-L` option of the `nfsmerge`(8) command for details).

An exceptions file can be used to prevent user names from being mapped. You can use a name that is not present in the remote `passwd` or `group` file as an exception for each name that is to be restricted. For example, assume that NFS access to a Cray system is to be restricted for login names `maleficent` and `stepmother` from the `disney` NIS domain. Entries for these users could be put into the exceptions file, as follows:

```
maleficent      BogusUser
stepmother      BogusUser
```

The login name `BogusUser` is not a valid login name in the `disney` NIS domain or on the Cray system. Therefore, user IDs for these users are not mapped between `groucho` and any machines using the `disney` NIS domain.

Examine the `passwd` and `group` files for exceptions before running the `Merge.domains` script, which is written to expect an exceptions file for all mappings. If an exceptions file is not present, but specified on the command line, the `Merge.domains` script issues a warning message.

3.1.6.4.5 Map Files

The previous `Get.domains` and `Merge.domains` example scripts assume that ID map files are maintained in the `/etc/uidmaps` directory.

The `Merge.domains` script creates map files only between the local Cray machine and all remote administrative domains. A log file, user ID file, and group ID file are created each time `nfsmerge` is called in the script. These files are placed in the `/etc/uidmaps` directory.

The log files contain a line identifying the type of ID map, the names of the local and remote `passwd` files, and a list of all names for which IDs were mapped. The example script is written so that it is obvious from the name of the map file which user or group ID map was created.

The files involved in mapping user IDs between local machine `harpo` and the NIS domain `nfl` in the example are as follows:

File	Description
`/etc/uidmaps/users/passwd.harpo`	A copy of the local `passwd` file sorted on login name
`/etc/uidmaps/users/passwd.nfl`	A copy of the remote `passwd` file sorted on login name
`/etc/uidmaps/users/e.harpo.nfl`	A list of login name exceptions between the administrative domains `harpo` and `nfl`
`/etc/uidmaps/l.harpo.nfl`	A log file from the creation of the user and group ID map between `harpo` and the `nfl` NIS domain
`/etc/uidmaps/u.harpo.nfl`	The user ID map file between `harpo` and the `nfl` NIS domain

Similarly, the files involved in mapping group IDs between `harpo` and the `nfl` NIS domain are as follows:

File	Description
`/etc/uidmaps/groups/group.harpo`	A copy of the local `group` file sorted on group name
`/etc/uidmaps/groups/group.nfl`	A copy of the remote `group` file sorted on group name
`/etc/uidmaps/groups/e.harpo.nfl`	A list of group name exceptions between the administrative domains `harpo` and `nfl`
`/etc/uidmaps/l.harpo.nfs`	A log file from the creation of the group ID map between `harpo` and the `nfl` NIS domain

3.1.6.5 Kernel Map Manipulation Example

Each ID map is given a name to use for display purposes and to use with some commands related to ID mapping. A separate map should be created for each autonomously administered system. For example, a map might be created for each stand-alone mainframe system on the network; one map would be required for a network of workstations in a single NIS domain.

Mapping in the kernel is a two-step process. The first step involves determining the particular ID mapping domain, given an Internet address. The second step uses the ID mapping domain, the type of map operation (user or group), and the direction (into or out of the Cray system) to determine the effective user or group identifier. Both of these operations occur in the UNICOS kernel with NFS configured and are based on information inserted into the kernel by the `nfsaddmap`(8) and `nfsaddhost`(8) utilities. See the *UNICOS Administrator Commands Reference Manual*, for a complete description of these commands.

Hosts are grouped into ID mapping domains based on sets of *address*, *mask* pairs on `nfsaddhost` calls. Addresses can be specified in standard form (for example, 128.1.0.1), as network names (names are found in `/etc/networks`), or as host names (from `/etc/hosts`). Default masks for hosts and standard forms are all 1's; for network names, the default masks are 1's covering the network part of the address (see the example of masks in the `nfsaddhost`(8) command description).

User ID mapping is straightforward. Each user ID map entry contains the local user ID and groups list and the remote user ID and groups list. When an NFS request is sent to the Cray system and the Cray system is mapping IDs for client-side requests, the Internet address to which the request is sent determines the ID map to be used. The map is searched for the local user ID; if it is found, the remote user ID and groups list is used in that request. If the local user ID is not found in the ID map, the ID mapping domain is checked to see what should be done (the `nfsaddhost`(8) command has options that determine the action associated with a particular ID mapping domain). There are three choices:

- The value for a bad user ID (-1) can be returned. In this case, the request should be denied with an RPC authentication error. This is the default.
- The value for the user "nobody" (-2) can be returned. In this case, the user has access only to other-accessible files and directories on the server.
- The local IDs can remain unmapped and can be put into the request. Group ID mapping is even simpler. The group ID map tables are used only to map file attributes (that is, to map IDs associated with a file that is accessed through NFS). Each group map entry contains a local group ID to correspond to a remote group ID.

Kernel ID mapping tables and ID mapping domains are inserted by the `/etc/uidmaps/Set.domains` script, called out of the `/etc/nfsstart` script. The `Set.domains` script example corresponds to step 3 of configuring ID mapping domains, as described in Section 3.1.6.2, page 248. This script primarily uses three ID mapping commands, `nfsclear`(8), `nfsaddmap`(8), and `nfsaddhost`(8)). The `nfsclear`(8) command is called first to ensure that any previous ID mapping information in the kernel is cleared out. The `nfsaddmap`(8) utility is called to read a map file, previously created with the `nfsmerge`(8) command, into the kernel. The `nfsaddhost`(8) utility defines the ID mapping domains; it associates Internet addresses with the kernel ID maps to use for mapping.

A special domain name, `MAP_THRU`, is defined for systems known to share the local Cray system's user and name space. The `MAP_THRU` domain simply allows user and group identifiers to pass through, without modifying them in any way.

Note: Do not use `MAP_THRU` with UNICOS security because the security information that NFS requires with UNICOS security will be missing.

The `Set.domains` example performs configurations, as follows:

Note: Your `Set.domains` file should contain the loopback entry as shown in the example; it is required if you are running the automounter.

1. Cray hosts `groucho` and `chico` perform server ID mapping between them.
2. Cray host `chico` performs all ID mapping between Cray host `harpo` and itself.
3. Cray hosts `harpo` and `groucho` perform client ID mapping between them.
4. Cray hosts `groucho`, `chico`, and `harpo` perform both client and server mapping to all other hosts on the network. The `Set.domains` script is as follows:

```
% cat /etc/uidmaps/Set.domains

#
#       Set
#

HOST=‘hostname‘MAPS=/etc/uidmaps
CMDS=/etc/uidmaps

#
#       Reinitialize kernel ID mapping information.
#

$CMDS/nfsidmap -d
```

```
sleep 3

$CMDS/nfsclear
$CMDS/nfsaddhost -l loopback
#
#       Set ID maps and ID mapping domains between all of
#       the Cray machines on the network.
#
if ‘test $HOST = groucho‘then
#       Groucho does server ID mapping only for chico
        $CMDS/nfsaddmap  -u $MAPS/u.$HOST.chico
                         -g $MAPS/g.$HOST.chico  chico
        $CMDS/nfsaddhost -d chico -s -l chico-inet
        $CMDS/nfsaddhost -d chico -s -l chico-prod
        $CMDS/nfsaddhost -d chico -s -l chico-lsp
        $CMDS/nfsaddhost -d chico -s -l chico-hsx -u chico-hsx2

#       Groucho does client mapping only for harpo.
        $CMDS/nfsaddmap  -u $MAPS/u.$HOST.harpo
                         -g $MAPS/g.$HOST.harpo  harpo
        $CMDS/nfsaddhost -d harpo -c -l harpo-inet
        $CMDS/nfsaddhost -d harpo -c -l harpo-vme24
        $CMDS/nfsaddhost -d harpo -c -l harpo-vme26
        $CMDS/nfsaddhost -d harpo -c -l harpo-vme32
        $CMDS/nfsaddhost -d harpo -c -l harpo-lsp

elif ‘test $HOST = chico‘then
#       Chico does server mapping only for groucho.
        $CMDS/nfsaddmap  -u $MAPS/u.$HOST.groucho
                         -g $MAPS/g.$HOST.groucho  groucho
        $CMDS/nfsaddhost -d groucho -s -l groucho-inet
        $CMDS/nfsaddhost -d groucho -s -l groucho-lsp
        $CMDS/nfsaddhost -d groucho -s -l groucho-hsx -u groucho-hsx2

#       Chico does both client and server mapping to harpo.
        $CMDS/nfsaddmap  -u $MAPS/u.$HOST.harpo
                         -g $MAPS/g.$HOST.harpo  harpo
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-inet
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-vme24
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-vme26
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-vme32
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-lsp
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-hsx
```

```
elif ‘test $HOST = harpo‘then
#       Harpo does client mapping only to groucho.
        $CMDS/nfsaddmap  -u $MAPS/u.$HOST.groucho
                         -g $MAPS/g.$HOST.groucho  groucho
        $CMDS/nfsaddhost -d groucho -c -l groucho-inet
        $CMDS/nfsaddhost -d groucho -c -l groucho-lsp
        $CMDS/nfsaddhost -d groucho -c -l groucho-hsx -u groucho-hsx2

#       Harpo maps through to chico (MAP_THRU facility).
        $CMDS/nfsaddhost -l chico-inet
        $CMDS/nfsaddhost -l chico-prod
        $CMDS/nfsaddhost -l chico-lsp
        $CMDS/nfsaddhost -l chico-hsx -u chico-hsx2

else
        echo "don’t know how to set domains for $HOST"
        exit
fi
# The Cray machines necessarily do both client and server mapping for
# all of the rest of the machines in the network.
#
#       zeppo
#
$CMDS/nfsaddmap -u $MAPS/u.$HOST.zeppo -g $MAPS/g.$HOST.zeppo zeppo

$CMDS/nfsaddhost -d zeppo -c -s -l zeppo-inet

#
#       Workstation network (nfl YP domain)
#

$CMDS/nfsaddmap -u $MAPS/u.$HOST.nfl -g $MAPS/g.$HOST.nfl nfl

#
#  Networks in the nfl YP domain
#

$CMDS/nfsaddhost -d nfl -c -s -l afc-eastnet -u afc-westnet
$CMDS/nfsaddhost -d nfl -c -s -l nfc-eastnet
$CMDS/nfsaddhost -d nfl -c -s -l nfc-centralnet
$CMDS/nfsaddhost -d nfl -c -s -l nfc-westnet
```

```
#
#  Explicit hosts in the nfl YP domain
#

$CMDS/nfsaddhost -d nfl -c -s -l nfl-gate
$CMDS/nfsaddhost -d nfl -c -s -l afc-gate
$CMDS/nfsaddhost -d nfl -c -s -l nfc-gate
$CMDS/nfsaddhost -d nfl -c -s -l nfc-east-server
$CMDS/nfsaddhost -d nfl -c -s -l nfc-central-server
$CMDS/nfsaddhost -d nfl -c -s -l nfc-west-server
$CMDS/nfsaddhost -d nfl -c -s -l afc-east-server
$CMDS/nfsaddhost -d nfl -c -s -l afc-central-server
$CMDS/nfsaddhost -d nfl -c -s -l afc-west-server
$CMDS/nfsaddhost -d nfl -c -s -l superbowl-server
$CMDS/nfsaddhost -d nfl -c -s -l bears-inet
$CMDS/nfsaddhost -d nfl -c -s -l colts-prod
$CMDS/nfsaddhost -d nfl -c -s -l giants-inet
$CMDS/nfsaddhost -d nfl -c -s -l redskins-prod
$CMDS/nfsaddhost -d nfl -c -s -l broncos-inet
$CMDS/nfsaddhost -d nfl -c -s -l vikings-inet
$CMDS/nfsaddhost -d nfl -c -s -l niners-prod
$CMDS/nfsaddhost -d nfl -c -s -l raiders-inet
$CMDS/nfsaddhost -d nfl -c -s -l patriots-prod
$CMDS/nfsaddhost -d nfl -c -s -l saints-prod#
#       Workstation network (disney YP domain)
#

#$CMDS/nfsaddmap -u $MAPS/u.$HOST.disney -g $MAPS/g.$HOST.disney disney

#
#  Networks in the disney YP domain
#

$CMDS/nfsaddhost -d disney -c -s -l snowwhitenet
$CMDS/nfsaddhost -d disney -c -s -l junglebooknet
$CMDS/nfsaddhost -d disney -c -s -l bambinet

#
#  Hosts in the disney YP domain
#

$CMDS/nfsaddhost -d disney -c -s -l disney-gate
$CMDS/nfsaddhost -d disney -c -s -l disney-land
```

```
$CMDS/nfsaddhost -d disney -c -s -l disney-world

$CMDS/nfsaddhost -d disney -c -s -l snowwhite-server
$CMDS/nfsaddhost -d disney -c -s -l snowwhite-inet
$CMDS/nfsaddhost -d disney -c -s -l bashful-inet
$CMDS/nfsaddhost -d disney -c -s -l sleepy-inet
$CMDS/nfsaddhost -d disney -c -s -l sneezy-inet
$CMDS/nfsaddhost -d disney -c -s -l dopey-prod
$CMDS/nfsaddhost -d disney -c -s -l dopey-inet
$CMDS/nfsaddhost -d disney -c -s -l happy-inet
$CMDS/nfsaddhost -d disney -c -s -l grumpy-prod
$CMDS/nfsaddhost -d disney -c -s -l doc-inet
$CMDS/nfsaddhost -d disney -c -s -l doc-prod
$CMDS/nfsaddhost -d disney -c -s -l junglebook-server
$CMDS/nfsaddhost -d disney -c -s -l junglebook-prod
$CMDS/nfsaddhost -d disney -c -s -l mowgli-inet
$CMDS/nfsaddhost -d disney -c -s -l hista-prod
$CMDS/nfsaddhost -d disney -c -s -l sherkan-inet
$CMDS/nfsaddhost -d disney -c -s -l baloo-inet
$CMDS/nfsaddhost -d disney -c -s -l kinglouie-inet
$CMDS/nfsaddhost -d disney -c -s -l bakkera-prod

$CMDS/nfsaddhost -d disney -c -s -l bambi-server
$CMDS/nfsaddhost -d disney -c -s -l bambi-inet
$CMDS/nfsaddhost -d disney -c -s -l thumper-inet
$CMDS/nfsaddhost -d disney -c -s -l flower-inet

$CMDS/nfsidmap   -e
```

3.1.6.6 Other Administrative Considerations

When ID mapping is configured, all server activity makes use of it. Most NFS client systems pass the `root` user ID (0) for user identification on their mount requests; these values are also subject to ID mapping.

You can remove kernel ID maps and ID mapping domains through the use of the `nfsrmmap`(8) and `nfsrmhost`(8) commands. These commands, along with `nfsaddmap`(8), `nfsaddhost`(8), `nfsadduser`(8), and `nfsrmuser`(8), give you the ability to modify ID mapping at any time; it is not necessary to recompile anything to modify the ID maps. To view the currently defined ID mapping domains, use the `nfslist`(8) command.

The following is an abbreviated sample of the use of `nfslist`(8) on machine `groucho`:

```
groucho% nfslist
NFS ID Mapping is : ENABLED

NFS ID    NFS ID                        Lower         Upper
Map       Mapping                       Bound         Bound         Address
Name      Flags                         Address       Address       Mask

nfs       CLIENT  SERVER  BAD_ID        c0.09.23.00   c0.09.26.00  [ff.ff.ff.00]
                      Addr(dec)         Addr(hex)     Host Name
                      128.1.35.0        c0.09.23.00   afc-eastnet
                      128.1.36.0        c0.09.24.00   afc-centralnet
                      128.1.37.0        c0.09.25.00   afc-westnet

zeppo     CLIENT  SERVER  BAD_ID        c0.09.0e.5a   c0.09.26.5a  [ff.ff.ff.ff]
                      Addr(dec)         Addr(hex)     Host Name
                      128.1.14.90       c0.09.0e.5a   zeppo-inet

chico             SERVER  BAD_ID        54.00.cf.05   54.00.cf.05  [ff.ff.ff.ff]
                      Addr(dec)         Addr(hex)     Host Name
                      84.0.207.5        54.00.cf.05   chico

chico             SERVER  BAD_ID        5c.00.00.05   5c.00.00.05  [ff.ff.ff.ff]
                      Addr(dec)         Addr(hex)     Host Name
                      92.0.0.5          5c.00.00.05   chico-lsp

harpo     CLIENT          BAD_ID        54.00.cf.06   54.00.cf.06  [ff.ff.ff.ff]
                      Addr(dec)         Addr(hex)     Host Name
                      84.0.207.6        54.00.cf.06   chico-lsp

harpo     CLIENT          BAD_ID        5c.00.00.06   5c.00.00.06  [ff.ff.ff.ff]
                      Addr(dec)         Addr(hex)     Host Name
                      92.0.0.6          5c.00.00.06   chico-lsp
```

To remove an ID mapping domain, the options of `nfsrmhost`(8) must exactly match the definition of the domain. You cannot remove part of a domain. Also, to remove a kernel ID map, you must remove all ID mapping domains that reference that map.

You must also be aware of hosts that are running NFS but are not UNIX systems. For these hosts, the `Get.domains` script used in the example must be modified according to specific characteristics of the site's network. For example, the script run in the example in this section assumes that `passwd` and `group` files exist on the remote system. This is not necessarily true for hosts that are not UNIX systems. To create the map files necessary for mapping IDs through NFS, you must construct files in `passwd` and `group` format for any administrative domains that do not already have these files (that is, you must create synthetic `passwd` and `group` files). If you create synthetic `passwd` and `group` files for use with ID mapping, you must ensure that entries are present for user `root` and group `sys`, or whatever these entries are called on the local Cray system.

3.1.6.7 Running `pcnfsd` with NFS ID Mapping Control

Note: The use of `pcnfsd` is not supported with the Cray ML-Safe configuration of the UNICOS system.

If you have a PC NFS client, the `pcnfsd`(8) daemon runs on an NFS server. When a PC NFS client connects to a `pcnfsd`, the client prompts for a login name and password. After verifying the password for the given login name, the `pcnfsd` daemon passes a user ID and a groups list back to the PC NFS client. The PC uses the IDs it receives from `pcnfsd` for subsequent NFS requests to that NFS server.

The `pcnfsd` daemon on a Cray system can use NFS ID mapping, which makes PC NFS access more secure. After the password validation that `pcnfsd` performs is complete, the user ID map entry for that user is added to an ID map. Therefore, if `pcnfsd` on a Cray system is configured to use an ID map, only users whose passwords were actually validated through `pcnfsd` can access a Cray NFS server.

To use `pcnfsd` with NFS ID mapping, ensure that the following steps have been taken:

1. Create an ID map file for a Cray system to the same Cray system in the `Merge.domains` script, as follows:

   ```
   nfsmerge -u /etc/uidmaps/u.cray.cray -g /etc/uidmaps/g.cray.cray
   /etc/passwd /etc/passwd /etc/group /etc/group
   ```

   Note: If you are also setting up a special `MAP_THRU` NFS ID map (see Section 3.1.6.9, page 266), the `nfsmerge` command needs to be executed only once because `pcnfsd` ID mapping and special `MAP_THRU` ID mapping use the same ID map file.

2. Add that ID map to the kernel with an appropriate name in the `Set.domains` script. The following command adds an empty user ID map and a group ID map called `pcidmap` to the kernel:

   ```
   nfsaddmap -g /etc/uidmaps/g.cray.cray pcidmap
   ```

   The user ID map is empty so that `pcnfsd` can add user entries when the user's password validation succeeds.

3. Ensure that the network addresses of the PCs that will be accessing the NFS server on the Cray system are in an ID mapping domain that uses the ID map (called `pcidmap` in the previous examples), as follows:

   ```
   nfsaddhost -d pcidmap -c -s -l pc_addr1
   nfsaddhost -d pcidmap -c -s -l pc_addr2
              .   .   .   .
   nfsaddhost -d pcidmap -c -s -l pc_addrN
   ```

4. Start `pcnfsd` with the name of the user ID map file and kernel map name, as follows:

   ```
   pcnfsd -u /etc/uidmaps/u.cray.cray -m pcidmap
   ```

   The `pcnfsd` daemon does not remove entries that `pcnfsd` has added to this map. Therefore, until the system administrator resets the ID maps in the kernel by running the `Set.domains` script or until the system reboots, any user validated through `pcnfsd` has NFS access to a Cray system from the PC network addresses in the ID mapping domains that reference `pcidmap`.

3.1.6.8 Deciding When to Use ID Mapping

Kernel ID maps contain the following information for each local user ID:

- Default account ID (acid)
- Security information (minimum and maximum security level and valid security compartments)
- Pointer to a list of optional Kerberos authenticated Internet addresses
- Pointer to a list of client side `auth_kerb` validated structures
- Pointer to a list of server side `auth_kerb` validated structures

Following is a description of circumstances in which it is desirable and circumstances in which it is necessary for the Cray NFS server to access this information:

- When acids rather than user IDs are being used for disk accounting and / or file quotas.

  In this case, NFS ID mapping is desirable, but not necessary. Acids are unique to UNICOS and therefore are not passed across the network as part of the NFS protocol. When files are created on the NFS server through NFS, the acid given to the file is the acid in the user structure of the `nfsd` process that does the first write operation to the file. Because acids are not part of the credentials in the NFS request, the acid attached to any file created across NFS is the acid of the running `nfsd` process (`root`'s default acid). This defeats disk quotas and disk accounting based on acids. Because ID maps contain the user's default acid, the NFS server can use this information when ID mapping occurs. A user cannot change the acid in the ID maps.

- When UNICOS security is enabled (with or without the IP security option (IPSO) enabled).

  In this case, NFS ID mapping is necessary. When UNICOS security information is required on the Cray system, and is passed across the network (through IPSO), the NFS server must validate the NFS requests based on the security information for the user making the request. The ID maps contain such security information.

  > **Note:** `MAP_THRU` ID mapping domains do not contain the required security information for UNICOS security, and cannot be used.

- When file systems or directories have been exported with the `krb` (Kerberos authentication required) export option in the `/etc/exports` file.

  In this case, NFS ID mapping is necessary. Users are required to run the `nfsid` command from the NFS client machine to the Cray NFS server machine to gain access to those file systems that have been exported with the `krb` option. The information by which users have been validated through Kerberos from certain Internet addresses is kept in the ID maps.

  > **Note:** The `kerberos` and `krb` operands used with the `exportfs` command are not supported on the Cray ML-Safe configuration of the UNICOS system.

- When file systems or directories in the `/etc/exports` file have been exported by using the `exportfs` command with the `-o` option and the `kerberos` operand (`auth_kerb` RPC authentication required). In this case, NFS ID mapping is necessary.

Following is a description of circumstances in which it is necessary for a Cray NFS client to access information contained in ID maps.

- When the mount(8) command with the -o option and kerberos operand (auth_kerb RPC authentication required) is used to mount an NFS file system. In this case, NFS ID mapping is necessary.

3.1.6.9 Special MAP_THRU NFS ID Map

If you need access to the information kept in ID maps (see the circumstances listed in the previous section), you must create a special ID map called a MAP_THRU map if one of the following situations is true:

- You have been running with NFS ID mapping and you have MAP_THRU ID mapping domains.
- You were not previously running with NFS ID mapping. In this case, you must also set up MAP_THRU ID mapping domains for all hosts and networks that are using the Cray system as an NFS server or client.

Typically, MAP_THRU ID mapping domains do not use a kernel ID map. However, if the MAP_THRU ID map is defined in the kernel, all MAP_THRU ID mapping domains use it. The special MAP_THRU ID map is built from the /etc/passwd file and /etc/group file from the local machine only. Following is a sample nfsmerge command to be added to the Merge.domains script that builds the ID map file:

```
nfsmerge -u /etc/uidmaps/u.cray.cray -g /etc/uidmaps/g.cray.cray /etc/passwd
/etc/passwd /etc/group /etc/group
```

The ID map file built by this command is the same ID map file that can be used with pcnfsd as described in Section 3.1.6.7, page 263). Therefore, if you are running pcnfsd with NFS ID mapping and you are using the special MAP_THRU NFS ID map, you need to execute the nfsmerge command only once.

To add the special MAP_THRU ID map to the kernel, add the following to the Set.domains file:

```
/etc/uidmaps/nfsaddmap -M /etc/uidmaps/u.cray.cray
```

To determine whether the MAP_THRU ID map is defined in the kernel, use the following command:

```
nfsidmem -v | grep MAP_THRU
```

If there are MAP_THRU ID mapping domains defined when the MAP_THRU ID map is loaded into the kernel, those ID mapping domains are also converted to use the MAP_THRU ID map. Conversely, if the MAP_THRU ID map is removed from the kernel, all MAP_THRU ID mapping domains (which point to the

MAP_THRU ID map) are converted back to standard MAP_THRU ID mapping domains (which do not point to an ID map). The special MAP_THRU ID map is the only ID map that can be removed from the kernel with the nfsrmmap command while ID mapping domains are referencing it. The following command adds the host address to the ID mapping domain:

```
/etc/uidmaps/nfsaddhost -l hostname
```

3.1.7 Configuring NFS Parameters

You can change NFS configuration parameters, such as the size of the rnode table, in several ways. All configurable NFS parameters appear in the /usr/src/uts/cf. *xxxx* /config.h file; you can change them by editing this file and building a new kernel. You can change the NFS parameters at boot time by entering appropriate entries in the network section of the system parameter file. You can use the UNICOS ICMS to make these changes, or you can make them manually.

3.1.7.1 Changing the config.h File

Table 4, Configurable NFS parameters, lists and describes the configurable parameters in the config.h file:

Table 4. Configurable NFS Parameters

Name	Default value	Description
NFS_MAXDATA	32768	Maximum number of bytes of user data read or written.
NFS_NUM_RNODES	256	Number of NFS rnodes. Each active NFS file or directory requires an rnode.
NFS_PORTMON	0	If set as non-zero, then NFS clients are required to use privileged ports (ports < IPPORT_RESERVED) in order to get NFS services.
NFS_PRINTINTER	0	Time interval in tenths of seconds between service not responding messages appearing on the console.
NFS_STATIC_CLIENTS	8	Number of permanently allocated client handles. Each NFS request to a server requires a client handle.
NFS_TEMP_CLIENTS	8	Number of temporarily allocated client handles. These client handles are destroyed when freed.

Name	Default value	Description
`CNFS_STATIC_CLIENTS`	8	Number of permanently allocated client handles for sending requests to a server whose file system is mounted with the `cray` mount option.
`CNFS_TEMP_CLIENTS`	8	Number of temporarily allocated Cray NFS client handles.
`NFS_MAXDUPREQS`	1200	Number of entries in the duplicate request cache. This value should be large enough so that the request entry is still present when the first retry of that request arrives.
`NFS_DUPTIMEOUT`	3	Time interval in seconds after the original request, during which duplicate requests received by the server are not reprocessed.

3.1.7.2 Changing the NFS Parameter File

You can change the configurable NFS parameters at boot time by placing entries in the network section of the system parameter file, `/etc/config/param`. The parameter names are the same as in the `config.h` file, except that they appear in lowercase.

Following is an example of the network section of a system parameter file:

```
network {
        .
        .
        .
        .
        512 nfs_num_rnodes;
        16  nfs_static_clients;
        16  nfs_temp_clients;
        .
        .
        .
        .
}
```

3.1.8 General Security Concerns

Although UNICOS NFS is an excellent tool for sharing files between computer systems, it also makes the files on a server vulnerable to unauthorized access. A user with `root` access on a workstation and a knowledge of how UNICOS

NFS works can pretend to be any user on the network and thereby gain access to server files that would not otherwise be accessible to that user. For example, the `/etc/exports` file and the `/etc/mountd` process are convenient mechanisms for providing the information needed by legitimate NFS clients. However, this information can usually be obtained by other means, making it possible to bypass the access controls that the `/etc/exports` file provides.

ID mapping provides some additional security by restricting access to NFS to those network addresses specified in the ID maps.

Because the standard NFS protocol was not designed with the use of access control lists (ACLs) in mind, access across NFS to files that use ACLs can be denied unexpectedly. (This does not occur if you use the `cray` option of the `mount`(8) command when processing between two Cray systems.) The NFS client checks the UNICOS permissions and sends the request to the server based on those permissions. However, the NFS server checks the ACL entries and grants or denies the request according to the procedures provided in the description of the UNICOS security feature in *General UNICOS System Administration*.

An additional security concern is the execution of `setuid` programs. An individual with `root` permission on a workstation or fileserver can create a `setuid root` program that can then be executed on a UNICOS NFS client. The `nosuid` argument to the `-o` option on the UNICOS `mount`(8) command prevents this operation.

The basic security mechanisms in UNICOS NFS are as follows:

Security mechanism	Description
Export control	Administrators can choose to restrict the list of hosts allowed to mount Cray file systems through the `exports`(5) file.
Mount control	Administrators control both the remote systems from which they import file systems and the permissions used on the UNICOS directories on which mounting is done through `mount`(8) options.
Standard UNICOS file permission checking	User and group ownerships and read, write, and execute-search permissions operate the same way on UNICOS NFS file systems as they do on UNICOS file systems. Users and administrators concerned about security should fully understand these mechanisms.

Kerberized NFS	Each NFS request is sent using the AUTH_KERB flavor of RPC. This RPC flavor protects packets by including an encrypted time stamp and other information on each packet. Users must have a valid Kerberos granting ticket prior to making NFS AUTH_KERB transactions.

Another level of security can be implemented through the ID mapping facility of UNICOS NFS. Administrators who elect to make exported file systems globally accessible (in the `/etc/exports` file) can impose restrictions through selective inclusion of remote addresses in the ID mapping domains, and they can further restrict access on those systems by the inclusion or exclusion of users in a particular domain's user or group ID map. (See Section 3.1.6.4.4, page 253, for more details.)

For information regarding avoiding mandatory access control (MAC) violation risks in the creation and use of NFS ID maps, see Section 3.1.6.4.3, page 252.

3.1.8.1 NFS and UNICOS Security

Note: This section describes NFS in a UNICOS security environment; however, no additional considerations exist for NFS with a secure environment, except that if you are running the Cray ML-Safe configuration of the UNICOS system, you must enable the Internet Protocol Security Option (IPSO).

When you are running UNICOS with UNICOS security, information about the sensitivity label of client users and server files must be communicated between the client and server hosts. No provision exists for this communication in the NFS protocol that UNICOS uses. To solve this problem, UNICOS places an interpretation on the labels of the datagrams that contain the NFS requests and responses.

When an NFS request is required, the client sends it at the label of the process that is attempting to access the file. The server uses the label of the request to perform mandatory access checks. The label must be a valid label for both the client host (through the NAL), and the user making the request. Valid user label ranges are stored in the kernel NFS ID maps. For this reason, ID mapping is required on UNICOS systems that run UNICOS security. See Section 3.1.6, page 246, for information on setting up ID maps.

After the server processes the request, the response packet is sent labeled with the sensitivity label of the file being accessed. If this label is invalid for the client host, the response is dropped.

When the client receives a response from the server, the label on the datagram is used as the sensitivity label of the file for any mandatory access checks that the client NFS software performs.

It is important to note that this scheme works only if both the client and the server can unambiguously determine the labels on datagrams passed between them. They can do this only if the systems use IP security labeling or the hosts are single-label hosts. If you are running with the `NETW_STRICT_B1` kernel option, NFS access is supported with any system that uses IP security options.

Warning: If you are running UNICOS security without the `NETW_STRICT_B1` kernel option (thus allowing multilabel ranges for hosts that do not use IP security options), do not export file systems to or mount file systems from any host that does not use IP security options and has a multilabel range in the NAL. UNICOS NFS is unable to correctly enforce MACs with such a configuration.

Note: Do not hard-mount file systems if one of the following is true:

- The Cray system is a system with UNICOS security and an NFS client.
- The NFS client system is mounting from a Cray NFS server system with UNICOS security.

With UNICOS security, requests that fail MAC do not receive responses; therefore, NFS requests on hard-mounted file systems hang.

Parameters in the `config.h` file that are supported with UNICOS security and NFS are as follows:

`NFS_SECURE_PORTMON`	Enables (nonzero) or disables (zero) whether NFS clients are required to use privileged ports (`ports < IPPORT_RESERVED`) in order to get NFS services. This, along with `NFS_PORTMON` (see Table 4), must be disabled for non-privileged access on MLS systems.
`NFS_SECURE_EXPORT_OK`	When set to a nonzero value, this parameter allows labeled file systems to be exported (used by the NFS server).
`NFS_REMOTE_RW_OK`	When set to a nonzero value, this parameter allows a Cray NFS client to mount a remote file system in read-write mode. When set to 0, NFS mounts the file system in read-only mode.

Note: If you are running NFS with `SECURE_MAC` enabled, the address associated with the `localhost` interface must be defined as `syslow` to `syshigh` levels and all compartments are in the Network Access list (NAL) (see `spnet`(8)). If `SECURE_MAC` is not enabled, the address associated with the `localhost` interface must be defined as 0 through 16 and all compartments are in the NAL. Under the Cray ML-Safe configuration of the UNICOS system, `SECURE_MAC` is enabled.

3.1.8.2 Kerberos Authentication

Note: Kerberos authentication is not supported with the Cray ML-Safe configuration of the UNICOS system.

Kerberos authentication can be required for NFS access to exported UNICOS file systems through the `krb` operand of the `exports`(5) command. This export option requires users to register with `mountd`, using the `nfsid`(1) command on the client machine from which they want NFS access to exported UNICOS file systems. ID maps are required to support the `krb` export option and are described in Section 3.1.6, page 246. For more information, see the `nfsid`(1) command.

The following examples show the mapping option (`-m`), which registers the user with `mountd` and the unmapping option (`-u`), which removes the user's registration:

```
nfsid -m remote_host_name
nfsid -u remote_host_name
```

Users who have not executed the `nfsid` command are granted only `others` access to files in file systems exported with the Kerberos option.

3.1.8.3 Kerberized NFS

Kerberized NFS uses the `AUTH_KERB` kernel RPC for NFS requests. Each NFS request contains additional information, which is validated by the NFS server's kernel.

UNICOS file systems may be exported by using the `exportfs`(8) command with the `-o` option and the `kerberos` operand. These exported file systems are mounted by using the `mount`(8) command with the `-o` option and `kerberos` operand. The super user, or user as root, must have an unexpired Kerberos Ticket Granting Ticket (TGT) before executing the `mount` command. A TGT is obtained by using the `kinit`(1) command.

Note: The `krb` and `kerberos` operands of the `exportfs` command may not be used at the same time.

Machines running Kerberized NFS must have an NFS principal entry in the `/etc/srvtab` file. Your Kerberos database must contain a principal of `nfs` and an instance of `hostname`. You must install a new `/etc/srvtab` file with the `nfs` principal prior to running Kerberized NFS.

The following guidelines must be understood in order to run Kerberized NFS:

- Your Kerberos database must contain specific information. For example, if your host name is `harpo`, you must have a principal of `nfs` and an instance of `harpo` in your Kerberos database. The `srvtab` entry would list `nfs` as the service and an instance of `harpo`.
- The `kerbd`(8) daemon must be running. `kerbd` handles requests from kernel level NFS and sends the requests to and from the Kerberos key distribution center (KDC).
- The user must have an unexpired TGT prior to attempting access to a Kerberos NFS mounted file system.
- A root user ticket must be regenerated every 21 hours. This is due to a limitation in the current implementation. All file systems mounted with the automounter and Kerberos NFS mount options are affected by this limitation.

3.1.9 UDP Checksum

The standard NFS implementation does not calculate user datagram protocol (UDP) checksums for the packets exchanged between NFS clients and servers. However, situations occur in which checksumming might be desirable, such as when the network is suspected to be error prone. Therefore, on UNICOS systems, checksumming for NFS is implemented in two parts: client and server.

To enable client-side checksumming, use the `cksum` argument on the `mount`(8) command. This argument causes the client to calculate and verify the checksums for all UDP packets sent to the server of this file system. However, this does not ensure that the server will also calculate and verify the checksum. You should confirm that the server in question verifies incoming checksummed packets.

The UNICOS NFS server automatically calculates and verifies the checksum for incoming checksummed packets. To enable server-side checksumming for outgoing packets, use the `cksum export` option within the `/etc/exports` file. However, this argument does not ensure that the NFS client that receives the packet will calculate and verify the checksum.

3.2 Troubleshooting

When a network service is not performing properly, the trouble usually lies in one of the following areas (listed from most likely to least likely):

- The network access control policies do not allow the operation, or architectural constraints prevent the operation.
- The NFS client software or environment is malfunctioning.
- The NFS server software or environment is malfunctioning.
- The network between the NFS server and client is malfunctioning.

The following sections offer a checklist for determining the location of the problem, some common problems, and a list of `mount`(8) command error messages.

Before trying to debug UNICOS NFS, read the man pages from the following lists that are relevant to your NFS environment:

Table 5. NFS man pages

Server	Client	ID mapping
`mountd`(8)	`mount`(8)	`nfslist`(8)
`nfsd`(8)	`automount`(8)	`nfsidmem`(8)
`exports`(5)	`biod`(8)	`nfsuid`(8)
`exportfs`(8)	`fstab`(5)	`nfsckhash`(8)
`portmap`(8)	`portmap`(8)	`nfsmerge`(8)
`rpcinfo`(8)	`rpcinfo`(8)	`nfsidmap`(8)
`pcnfsd`(8)	`nfsid`(1)	`nfsaddmap`(8)
		`nfsaddhost`(8)
		`nfsclear`(8)
		`nfsadduser`(8)
		`nfsgid`(8)
		`nfsrmhost`(8)
		`nfsrmmap`(8)

3.2.1 Isolating the Problem

The following sections contain a checklist to help you either resolve the problem or isolate the problem (that is, help identify the environment in which the problem occurred). This checklist is sequential, and it verifies whether all of the basic functions required for NFS are working. Use this checklist as a starting point if you have no idea where the NFS problem is occurring.

The checklist is grouped into the following topics:

- NFS mounting problems
- Problems accessing NFS mounted files
- Problems with ID mapping

If you are having problems configuring NFS for the first time, use the complete checklist. Individual items within the checklist can be used at any time to help isolate or resolve problems when NFS is already up and running.

This checklist assumes that an NFS client is having problems mounting or accessing NFS files from an NFS server.

3.2.1.1 NFS Mounting Problems

If the `mount`(8) command times out, perform the following steps:

1. Ensure that the NFS server machine is up and that you can access the NFS server from the NFS client. On the NFS client, use the `ping`(8) command, as follows:

   `ping` *server_hostname*

2. Ensure that the NFS server machine is at least running either `portmap`(8) or `rpcbind`(8), `mountd`(8), and `nfsd`(8) daemons. On the NFS server machine, use the `ps`(1) command, as follows:

   ```
   ps -ae | egrep 'portmap|rpcbind|mountd|nfsd'
   ```

   Following is sample output from the `ps` command:

   ```
   668   -    0:02 portmap
   1347  -    0:00 mountd
   1343  -    0:00 nfsd
   1341  -    0:00 nfsd
   1342  -    0:00 nfsd
   1344  -    0:00 nfsd
   ```

Only one `portmap` and `mountd` daemon should be running at any one time, but one or more `nfsd` daemons can be running at one time (a typical number is 4).

If more than one `mountd` daemon is running, conflicts regarding the `mountd` requests can occur and the client mount requests will not be serviced.

If `portmap` or `rpcbind`(8) is not running, stop the `mountd`, `nfsd`, and `pcnfsd`(8) daemons. You must also stop any other RPC registered servers that are running. Then start the `portmap` or `rpcbind` daemon. After this daemon is started (the `ps -ae` command shows `portmap` or `rpcbind` running), start `mountd` and the `nfsd` daemons.

Note: Look into server idiosynchracies, for example, if the NFS server is a Silicon Graphics (SGI) system and the Cray system is the client, the `mountd` daemon on the SGI system must be started with the `-n` option.

Sun Microsystems has the `-n` option set by default within their startup scripts; other systems may vary.

3. Ensure that `portmap`(8) or `rpcbind`(8), `mountd`(8), and `nfs`(4P) are registered RPC services on the NFS server machine. On the client, use the `rpcinfo`(8) command, as follows:

   `rpcinfo -p` *server_hostname*

   Following is partial sample output from the `rpcinfo` command:

```
program vers proto   port
100005    1   tcp    678  mountd
100005    1   udp    676  mountd
100003    2   udp   2049  nfs
        :
        :
        :
100000    2   tcp    111  portmapper
100000    2   udp    111  portmapper
```

   There are two entries for `mountd` and `portmapper`, one for `udp`, and the other for `tcp`. NFS has only one entry, because it uses only `udp` as the transport protocol.

   If one or more of these programs are not registered RPC programs, either step 2 has failed, or you should kill the `portmap` or `rpcbind`, `mountd`, and `nfsd` daemons (along with any other RPC registered programs) and restart them.

Start `portmap` or `rpcbind` first; after it has begun, start the `mountd` and `nfsd` daemons, along with any other RPC daemons.

The `-u` option of the `rpcinfo` command can also be run on the client to determine whether these servers are registered and responding through `portmap`. The `-u` option uses user datagram protocol (UDP) to the specified server on the specified program, as in the following example:

```
rpcinfo -u server_hostname 100005 1
```

The program number for `mountd` is 100005.

Following is sample output from the `rpcinfo` command:

```
program 100005 version 1 ready and waiting
```

4. Ensure that the file system or directory you are trying to mount is exported on the NFS server giving the client permission to mount. Use the following command only if you are running as `root`:

   ```
   /etc/exportfs
   ```

   The `exportfs` command without options prints the currently exported file systems or directories.

   If you are not running as `root`, use the following command:

   ```
   cat /etc/xtab
   ```

   This file is updated by `exportfs`; thus, it also shows the current list of exported file systems or directories.

   Note: Viewing the `/etc/exports` file does not necessarily show the currently exported file systems or directories; therefore, it should not be used to determine whether something is exported.

5. Ensure that the `/etc/hosts` files are accurate on both the NFS server and client. Check to verify that the entry in the `/etc/hosts` file for the server is the same on both systems (that is, the server's host name used on both systems points to the same Internet address), and perform the same check with the entry for the client on both systems.

   Note: If you are running the domain name server, using the `named`(8) daemon, the `/etc/hosts` file is not accessed and you should use either the `nslookup`(1) or the `host`(1B) command to identify the host entry. See the man pages for details. If you are using NIS, use the `ypcat HOSTS | grep` *host* command, where *host* is the name of the host or machine you want to access.

See the troubleshooting steps in Section 3.2.1.3, page 280, if you are using ID mapping and either the NFS server or client has multiple network interfaces (therefore, multiple `/etc/hosts` file entries).

3.2.1.2 Problems Accessing NFS Mounted Files

After the NFS system is mounted, do the following if you cannot access these files:

1. Ensure that the file system or directory is still mounted on the NFS client. Use the following `mount`(8) command without options to list all currently mounted local and NFS file systems:

   ```
   /etc/mount
   ```

   The `rsize` and `wsize` options of the `mount` command specify the number of bytes in the read buffer and the write buffer, respectively (they are typically set to the same value). Ensure that these options are correct.

   To determine these values, check the exact `mount` command used (you may need to look in `/etc/fstab` to determine the exact options used). If an `rsize` or `wsize` option is not used with the `mount` command, these values are set to a default size. On UNICOS systems, the default size is 8 Kbytes. However, the maximum size, 32 Kbytes, is set by the `NFS_MAXDATA` kernel variable in the `config.h` file. A similar variable should exist on systems that are not UNICOS systems. Contact the appropriate vendor for this information.

   Following are some suggestions for setting `rsize` and `wsize` on the `mount` command under various configurations:

   - Cray-to-Cray NFS environment

     For UNICOS systems, the default value for read and write buffers is 8 Kbytes, which is the maximum buffer size for many other systems. However, if you are running UNICOS NFS between two Cray systems, you should set `rsize` and `wsize` to the maximum value of 32 Kbytes. With Cray systems running earlier releases of UNICOS, it is not necessary to specify `rsize` and `wsize`, because the default is 32 Kbytes.

   - Cray NFS client and other vendor's NFS server

     The `rsize` and `wsize` values set on the Cray NFS client, or the default value set if these options were not specified, must not exceed the maximum read or write buffer size of the NFS server. A typical maximum value for other vendors is 8 Kbytes. If you are running UNICOS 9.0 or later, the client default value is 8 Kbytes; therefore, it is not necessary to specify the `rsize` and `wsize` options, unless the NFS server's limit is less than 8 Kbytes.

Contact your vendor for the maximum buffer size (`NFS_MAXDATA`) supported. See Section 3.4, page 293, for more details.

- Cray NFS server and other vendor's NFS client

  In this configuration, the limiting factor is again the other vendor's maximum buffer size (`NFS_MAXDATA`). See Section 3.4, page 293, for more details on `rsize` and `wsize`.

2. Ensure that the access options and the directory and file permission settings on the NFS server and client are not the problem.

   On the NFS server, check the following `exportfs`(8) or `exports`(5) options to see whether the file system or directory was exported with any options that affect permissions:

Option	Description
`ro`	Export read-only.
`rw=` *hostname*	Export read-write to *hostname* only.
`anon=` *uid*	User ID for unknown user.
`root=` *hostname*	Give `root` access to *hostname* only.
`krb`	Kerberos authentication required for access.

   See the `exportfs`(8) and `exports`(5) man pages for more detail.

   On the NFS client, check the permissions of the following:

   - The mount point. If the file system or directory is already mounted, you must unmount it to obtain the actual mount point directory permission settings. At a minimum, the mount point directory permissions should be set to 555.

   - The mounted files or directories that are having trouble being accessed. Check the owner and group name and permission settings to see whether this might be the problem.

     Ensure that you have a flat administrative environment in which all user IDs are the same across all systems, or that you are running NIS, or that you have UNICOS ID mapping set up and running. See the following section for more information on ID mapping troubleshooting). If you are not using UNICOS ID mapping (that is, you do not have map-through

ID mapping domains), ensure that ID mapping is disabled by using the `/etc/uidmaps/nfsidmap -d` command. You must have `root` permission to use this command.

3. Ensure that enough memory buffers (mbufs) are available. Use the following command to obtain an mbuf count:

   ```
   /etc/netstat -m
   ```

   See Chapter 2, page 3, and `netstat`(1) for more details on mbufs and their use.

3.2.1.3 Problems with ID Mapping

If the Cray system is an NFS server or NFS client, UNICOS ID mapping can occur. It is recommended that if both server and client are Cray systems, ID mapping should occur on the NFS server. You can perform the following troubleshooting steps on either an NFS server or client, depending on where the ID mapping is occurring.

1. Ensure that ID mapping is enabled. Output from the following command (without options) specifies whether ID mapping is enabled or disabled:

   ```
   /etc/uidmaps/nfsidmap
   ```

2. Ensure that an ID mapping domain exists for the system experiencing problems. Use the following command to list the ID mapping domains that contain the *hostname* address:

   ```
   /etc/uidmaps/nfslist -a hostname
   ```

   The host name specified in this command should be the same host name specified on the `mount`(8) command.

   If multiple network interfaces (that is, multiple network paths) are between the NFS server and client systems, ensure that an ID mapping domain is set up for the correct interface or for all such interfaces, if routing is not static.

3. Ensure that ID maps and corresponding hash tables are accurate. Use the following command (without options) to check the consistency of the user ID and group ID maps in the kernel:

   ```
   /etc/uidmaps/nfsckhash
   ```

   If this command indicates any problems, follow the procedure to rebuild your ID maps, as previously described.

4. Ensure that the user(s) are set up correctly. During the actual `mount` command operation, `root` is the user. If the Cray system is experiencing access problems when trying to mount an NFS file system, `root` is the affected user. You should not have any exceptions listed in an exceptions file for user `root`. For example, although `root` must be present in an exceptions file for mounting, you should not attempt to map `root` to the bad user ID.

   After the file system is mounted, any user can try to access that file system. Ensure that the user entries, such as group lists, are in the required ID map and that the entries contain the most current information. Use the following command to print the user entries:

   `/etc/uidmaps/nfsuid -m` *map_name list_of_user_names*

   The specified map name should be the map listed as output from the `nfslist` command described in step 2.

   If a map-through map was created for the domain in question and you are not running UNICOS security, you do not need to perform this command.

5. Because the UDB allows user entries with no group ID, it creates password files with empty group ID fields. Because the `nfsmerge`(8) command does not accept an entry without a group ID field, you might receive the following message if you execute `nfsmerge`:

   ```
   nfsmerge: WARNING!! Could not read entire password file.
   There is probably an invalid entry (no gid?).
   The resulting ID map file may be incomplete.
   ```

   For mapping to work, a user must have a group ID. If this error occurs, assign the user a group ID.

3.2.2 NFS Mount Failure

This section consists of an example of an NFS mount, followed by a list of error messages with explanations. If your `mount`(8) command fails for any reason, check the generated error messages for information about possible solutions.

3.2.2.1 NFS Mount Example

The `mount`(8) command can get its parameters from the command line or from the `/etc/fstab` file. The following example assumes command-line arguments, but the same debugging techniques work if `/etc/fstab` is used. Look at a sample mount request made from a client machine:

```
mount -t NFS krypton:/usr/src /krypton.src
```

The example asks the server machine called `krypton` to return a file handle for the `/usr/src` directory. This file handle is then passed to the kernel in the `mount`(2) system call. The kernel looks up the `/krypton.src` directory; if everything is working properly, it ties the file handle to the directory in a mount record. From now on, all file system requests to that directory, and any subdirectories, will go through the file handle to `krypton`.

The following is a list of steps the `mount` command takes to mount a remote file system, as in the previous example:

1. The `mount`(8) command parses the first argument into host `krypton` and remote directory `/usr/src`.
2. The `mount` command determines the Internet Protocol (IP) address of `krypton`.
3. The `mount` command calls the `krypton mountd`(8) program and passes `/usr/src` to it.
4. The `krypton` server's `mountd` command reads `/etc/exports` and looks for the exported file system that contains `/usr/src`.
5. The `krypton` server's `mountd` command expands the host names and network groups in the export list for `/usr/src`.
6. The `krypton` server's `mountd` command gets a file handle for `/usr/src` from the operating system.
7. The `krypton` server's `mountd` command returns *fhandle*.
8. The `mount`(8) command performs an NFS `mount` system call with the file handle and the `/krypton.src` directory.
9. The NFS `mount` system call determines whether the caller is a super user and whether `/krypton.src` is a directory.
10. The NFS `mount` system call does a `statfs`(2) system call to `krypton` 's UNICOS NFS server (`nfsd`).
11. The `mount` command opens the `/etc/rmtab` file and appends an entry to it.

3.2.2.2 NFS Mount Failure Error Messages

Any one of the steps in the previous section can fail, some of them in more than one way. Following are specific error messages, along with descriptions of the failures associated with each.

`/etc/fstab:  No such file or directory`

The `mount`(8) command tried to search for the name in `/etc/fstab`, but `/etc/fstab` did not exist.

`mount:  ...  Block device required`

A likely cause is the omission of the `krypton`: part of the following request:

```
mount krypton:/usr/src /krypton.src
```

The `mount`(8) command assumes that you are doing a local mount, unless there is a colon in the file system name or the file system type is NFS in `/etc/fstab`.

`mount:  directory path must begin with /`

The second parameter to `mount` identifies the path of the specified directory. This must be a full path name beginning with `/`.

`mount:  ...:  No such file or directory`

Either the remote directory or the local directory does not exist. Check the spelling, and use `ls`(1) to request a listing of each directory.

`mount:  ...:  Not a directory`

Either the remote path or the local path specified is not a directory. Check the spelling, and use `ls`(1) to request a listing of each directory.

`mount:  ...  not found in /etc/fstab`

If `mount` is called with either a directory or a file system name, but not both, it looks in `/etc/fstab` for an entry whose file system or directory name field matches the argument on the command line. For example, the following entry results in a search of `/etc/fstab` for a line that has a directory name field of `/krypton.src`:

```
mount /krypton.src
```

Assume that an entry such as the following is found:

```
krypton:/usr/src /krypton.src NFS rw,soft,rsize=8192,wsize=8192
```

The mount is then performed as though you had typed the following:

```
mount -t NFS -o rw,soft,rsize=8192,wsize=8192 krypton:/usr/src /krypton.src
```

The "not found" message indicates that a match for the argument given to `mount` was not found in any of the entries in `/etc/fstab`.

`mount:  not in export list for ...`

In the `/etc/exports` file, your machine name is not included in the export list for the file system you want to mount. You can look at the file by logging in to the server system or by using `remsh`(1B). If you are to mount the system, add your machine names to the relevant exports list, or the list of machine names should be null, indicating that any machine can mount the file system.

`mount:  ...:  Not owner`

You must do the mount as super user on your machine because it affects the file system configuration for the whole machine, not just for you.

`mount:  ...:  Permission denied`

This message generally indicates that some authentication failed on the server. It could simply be that your machine name is not included in the appropriate `/etc/exports` list (see the preceding message) or that the server could not determine who you are. Possibly, the server does not acknowledge that you are who you say you are. Check the server's `/etc/exports` file.

`mount:  ...  server not responding:  RPC_PMAP_FAILURE - RPC_TIMED_OUT`

Either the server from which you are trying to mount is down, or its portmapper is dead or hung. Try logging in to that machine. If you can log in, enter the following:

```
rpcinfo -p hostname
```

You should see a list of registered program numbers. If you do not, you might have to restart the `portmap`(8) daemon. If you cannot perform a remote login to the server, but the server is up, you should check your network connection by trying a remote login to some other machine. You should also check the server's network connection.

`mount:  ...  server not responding:  RPC_PROG_NOT_REGISTERED`

This message indicates that `mount` got through to the portmapper, but the NFS mount daemon (`mountd`) was not registered.

`arp;... unknown host`

The host name you supplied could not be found in `/etc/hosts`. First check the spelling and the placement of the colon in your `mount` call. If the spelling and syntax are correct, try `ping`(8) on the local machine (client) and on another machine to determine whether the remote host is responding.

For UNICOS, `mountd`(8) is typically started from `/etc/netstart`. Use `sdaemon`(8) to start it manually. Generally, it can be restarted by simply entering the following command:

```
/etc/mount mountd
```

3.2.3 Hanging Programs

If programs hang while they are performing file-related work, your server might be dead. In this case, you might see the following message on your console:

`NFS server` *Internet address* `not responding, still trying`

This problem originates with either one of your servers or the network. If the problem is with a server, you can determine which server is malfunctioning by locating the address provided in the message in `/etc/hosts`. If your machine hangs completely, check the server(s) from which you have mounted. If one (or more) of them is down, your programs will continue automatically when the server comes back up. There will be no indication that the server died, and no files will be destroyed.

If a soft-mounted server dies, other work should not be affected. Programs that time out while trying to access soft-mounted remote files will fail with the `errno` value of `ETIMEDOUT,` but you should still be able to work on your other file systems.

If all servers are running, check to see whether anyone else using the server or servers in question is having trouble. If more than one machine is having problems getting service, it is probably a problem with the server's daemon (`nfsd`(8)). Log in to the server and execute the `ps`(1) command to see whether `nfsd` is running and accumulating CPU time. If not, you might be able to terminate and then restart `nfsd`. If this does not work, you must reboot the server.

If no one else is having problems, check the network connection and the connection of the server.

If your machine comes up partially after a boot, but it hangs where it would usually be doing NFS mounts, one or more servers is probably down or a problem exists with your network connection.

3.2.4 No Super-user Access over the Network

Under UNICOS NFS, a server exports the file systems it owns so that clients can mount them remotely. When you become a super user on a client, you are denied access on remotely mounted file systems. Consider the following example:

```
% cd
% touch test1 test2
% chmod 777 test1
% chmod 700 test2
% ls -l test*
-rwxrwxrwx  1 jsbach    0 Mar 24 16:12 test1
-rwx------  1 jsbach    0 Mar 24 16:12 test2
```

Now, retry it as super user.

```
% su
Password:
# touch test1
# touch test2
touch: test2: Permission denied
# ls -l test*
-rwxrwxrwx  1 jsbach    0 Mar 21 16:16 test1
-rwx------  1 jsbach    0 Mar 21 16:12 test2
```

The problem usually appears during the execution of a `setuid root` program. Programs that run as `root` cannot access files or directories, unless the permission for `other` allows it.

Also, if the server is not an NFS server and the `export`(5) file or the `export` option on the `exportfs`(8) command does not allow your workstation `root` access, you cannot change ownership of remotely mounted files. Because, in this case, users cannot perform a `chown`(1) command, and the super user is treated as a standard user on remote access, no one but the super user on the server can change the ownership of remote files. For example, if you try to execute `chown` as yourself on new program `a.out`, which must be `setuid root`, it will not work, as demonstrated in the following example:

```
% chmod 4777 a.out
% su
Password:
# chown root a.out
a.out: Not owner
```

To change the file ownership, you must log in to the server as a super user and then make the change. Alternatively, you can move the file to a file system owned by your machine (for example, `/usr/tmp` is always owned by the local machine) and make the change there.

3.2.5 File Operations Not Supported

Remote file systems support file locking if the daemons `lockd`(8) and `statd`(8) are running. By default, file locking is supported.

If you do not want file locking, mount the file system using the `nolock` option on the `mount`(8) command.

Append mode and atomic writes are also not ensured to work on remote files that are accessed simultaneously by multiple clients.

3.2.6 Remote Device Access Not Supported

Under UNICOS NFS, you cannot access a remotely mounted device or any other character or block special file, such as a named pipe.

3.3 Confidence Testing

The UNICOS NFS confidence test suite is provided to ensure the proper installation of UNICOS NFS. These tests are used by system analysts and administrators. Two types of test groups are included in the test suite: functional and performance.

Functional tests verify that specific features are working as expected. If the functional tests fail, you should look for errors in installation or configuration. The functional test group includes the following groups of tests:

Test group	Description
Basic	Determines whether UNICOS NFS is providing basic functionality
General	Checks some commonly used applications such as compiles, `nroff`(1), and `tbl`(1)
Special	Analyzes specific facilities that have required special attention in the past
Cray client	Analyzes facilities that have required special attention on Cray systems as NFS clients

After the functional tests pass, you can look at performance to determine whether the system can be fine-tuned to run faster. The performance test provides file transfer rate measurements.

All confidence tests are self-checking. Error messages are provided to help you determine the source of a problem. The tests described in this section are designed to be compiled and executed on client machines running the UNICOS operating system. No test suite software runs on the server.

During the execution of the test suite on a client machine, directories and files are created along a separate path called the *test directory*. The test directory usually consists of the path to a remote directory that was mounted on the client from a server. Optionally, the test suite can mount a remote directory on the local machine.

3.3.1 Installation

The `/usr/src/net/nfs/tests` directory contains the confidence test suite source code. This source code should be copied to a user directory before it is compiled. The `/usr/src/net/nfs/tests/Makefile` makefile can be used to distribute the test source code to a new location, as follows:

```
su root
make dist DESTDIR=destination_path_for_test_source
```

After placing the source tree in the desired location, use the `Makefile` makefile to compile the tests. Execute the following command in the destination directory:

```
cd destination_path_for_test_source
make all
```

To move the compiled tests to a new location, use the following makefile:

```
su root
make copy DESTDIR=destination_path_for_compiled_tests
```

When the test suite is created successfully and is in its execution location, the permission mode and ownership of binary file `domount`, which exists in the root of the test suite tree, must be changed. The super user must own `domount`, and the `setuid` bit must be set. As super user, enter the following commands:

```
chown root domount
chmod 4555 domount
```

3.3.2 Test Execution

Execute the test suite as follows:

1. As super user, mount the desired NFS directory.

   Example:

```
mount -t NFS -o rw,soft,wsize=8192,rsize=8192 cray2:/tmp /usr/tmp/mount
```

2. As a standard user, set the `NFSTESTDIR` environment variable to a subdirectory of the mounted directory. For example, use the mount point in the previous example, as follows:

   ```
   setenv NFSTESTDIR /usr/tmp/mount/craytest
   ```

   If the test directory already exists, the test suite deletes it and its contents (`craytest`, in this example).

3. Modify the `TESTS` and `TESTARG` variables in the `tests.init` file to indicate the type of functional test you want to run, as follows:

   - Set the `TESTS` variable to indicate which of the following tests you want to run:

Variable	Test type
`TESTS="-b"`	Runs the basic tests
`TESTS="-g"`	Runs the general tests
`TESTS="-s"`	Runs the special tests
`TESTS="-c"`	Runs the Cray client tests
`TESTS="-a"`	Runs all of the preceding tests (default)

- Set the `TESTARG` variable (which affects only the basic tests) to run an abbreviated functional test, as follows:

Variable	Test type
`TESTARG="-f"`	Causes the basic tests to run a shorter functional test. Use this variable in conjunction with `TESTS="b"` to run a quick-look test of NFS.
`TESTARG="-t"`	Causes the basic tests to run a longer test with timing functions (default).

- Run the functional tests with the `runtests` command, as follows:

```
runtests
```

The `runtests` script (located in the root directory of the test tree) is invoked on the client machine. If the test directory exists, you are given the following message and choice:

```
The NFSTESTDIR directory /usr/tmp/mount/craytest exists.
The NFS tests expect to create this directory.
Remove the existing directory (and its contents!)
and continue with the NFS tests (n[y])?
```

To delete the directory and continue, type **y**; to abort the test and leave the directory intact, type **n**. `n` is the default.

- Run the performance test as a separate test. For meaningful performance numbers, the client machine, the network, and the server machine must be dedicated to the test. Run the test as follows:

```
nfsperf  [-l num_times_to_loop_thro_test]
         [-d results_directory]
         [-n num_of_concurrent_tests]
         [-s file_size]
         [-f num_of_files]
         testdir
```

Option	Description
`-l`	Determines the number of times to repeat each test; default is 1.
`-d`	Indicates that the `results_directory` directory is to be used to store the results files.

`-n`	Indicates the number of concurrent tests; default is 1.
`-s`	Indicates the size of files that will be created during the tests in multiples of 32 Kbyte blocks; default is 50.
`-f`	Indicates the number of files to create for each test; default is 5.
testdir	Name of directory that contains the tests. This parameter is optional if the `NFSTESTDIR` environment variable has been set.

3.3.3 Test Configurations

The following test configurations are suggested for UNICOS NFS confidence testing in a Cray environment:

1. Run the test suite on a Cray machine, specifying `NFSTESTDIR` as a local (not remotely mounted) directory. The tests run against standard file space, providing a baseline set of results and timings (speed of the Cray file system) if the `-t` option is selected for the basic tests).
2. Make the Cray machine both the server and the client machine by mounting a local file system onto another local directory, using `localhost` as the *server_name* argument on the `mount`(8) command.
3. Mount a file system from a remote server machine onto the Cray system. The test directory then exists on a remote machine.

3.3.4 Executing Individual Tests

You can execute any individual test by setting the `NFSTESTDIR` environment variable to the name of a directory within the mounted directory. For example, if a remote directory is mounted on local test directory `/usr/tmp/mount`, specify a new subdirectory, such as `/usr/tmp/mount/craytest`. Then execute the following command:

```
setenv NFSTESTDIR /usr/tmp/mount/craytest
```

To run a basic test, change to the basic test directory, check the specific test for required arguments, and then run the test, as follows:

test_name   arguments

For example, to run functionality test `test3` from the basic section, execute the following commands:

```
cd basic
test3 -f
```

To run any of the tests other than the basic tests, you must first copy the tests to the mounted test directory.

Example:

```
mkdir $NFSTESTDIR
make copy DESTDIR=$NFSTESTDIR
cd $NFSTESTDIR
rename 100
```

3.3.5 Cleaning up

If the `runtests` script is used to run the test suite, all files and directories created in the test directory are removed at the completion of testing. However, the tester must clean up the test directory after running an individual test.

The following command attempts to remove files in the test suite, including `$NFSTESTDIR` :

```
rm -rf $NFSTESTDIR
```

The super user must remove executable file `domount`.

3.3.6 Test Contents

Basic tests create and remove files and directories, obtain and set file attributes, perform look-up functions, read and write files, read directory entries, and obtain file system statistics.

Special test functions include checking access to open files that have had their modes changed, checking replies lost on nonidempotent requests, performing exclusive create functions, performing seeking functions to a negative offset, repeatedly renaming files, creating and accessing files with holes (data blocks not allocated), and taking proper `umask`(1) action on remote files.

Performance tests provide a file transfer rate benchmark. Several files are created from the mounted file system, and then data is written to and read from those files.

General file system tests include compiles, simultaneous compiles, doing a `makefile`, nroffing a file, or using `tbl`. They also provide timing information for performance measures.

Cray client tests include tests of special areas that are unique to UNICOS, including asynchronous I/O, truncation tests, and additional `umask` tests.

Please send comments and suggestions concerning the UNICOS NFS test suite to the following:

Cray Inc.

Software Test Group

1340 Mendota Heights Road

Mendota Heights, MN 55121

Additional tests to be added to the suite are welcome.

3.4 Performance and Tuning

NFS is a synchronous protocol designed for reliable remote access. The synchronous characteristic means that for each NFS request sent, a response must be received (indicating the completion of the request) before another NFS request can be sent. This characteristic is one of the primary reasons NFS is not as fast as other TCP/IP applications. NFS uses UDP/IP, which is a connectionless, unreliable transport protocol (NFS does not use TCP). Therefore, NFS runs best over reliable local area networks.

Many NFS performance factors are related to the manner in which UNICOS TCP/IP networking parameters were tuned or optimized. Particularly affecting performance are maximum transmission units (mtus) (see Section 2.3.1, page 123); memory buffers (mbufs) (see Section 2.3.2, page 128); and network routing (see Section 2.3.3, page 148).

The following sections describe factors that affect NFS performance and methods for obtaining performance figures.

3.4.1 Factors That Affect NFS Performance

The following sections describe factors that can affect NFS performance and offer guidelines for increasing performance.

3.4.1.1 `NFS_MAXDATA` Parameter

The `NFS_MAXDATA` parameter (defined in the `config.h` file) defines the maximum size of the data part of a remote NFS request. By default, `NFS_MAXDATA` is set at 32 Kbytes (32,768 bytes) on UNICOS systems. This parameter has the greatest effect on NFS servers, but it also affects NFS clients. For example, a Cray system acting as an NFS server can receive a maximum of 32 Kbytes of data from an NFS client request. However, the maximum amount of data an NFS client can send is determined by the value of its `NFS_MAXDATA` variable. This makes the limiting factor the system with the smallest `NFS_MAXDATA` size. Generally, the more data that can be sent and received at one time, the better the performance. Therefore, it may be advantageous to increase `NFS_MAXDATA` on systems that are not Cray systems.

Note: It is not known how all other systems define this parameter, or whether the parameter can be changed on all of these systems. Contact the appropriate vendor for such information.

3.4.1.2 `mount` Command Arguments

The `mount`(8) arguments `rsize` and `wsize` can have a direct effect on NFS performance. You can set these parameters when issuing the `mount` command on the NFS client. However, `rsize` and `wsize` cannot be set above the client's `NFS_MAXDATA` size, and they should not be set greater than the server's `NFS_MAXDATA` value.

The `rsize` parameter specifies the read buffer size in bytes (the maximum amount of data the NFS client can accept from an NFS read request); the `wsize` parameter specifies the write buffer size in bytes (the maximum amount of data the NFS client will send in an NFS write request). Generally, `rsize` and `wsize` are set to the same value. A guideline to use in setting these parameters is to set them as large as possible, but not to exceed the smaller of the `NFS_MAXDATA` size of the client or server. If these mount parameters are not specified, the default used is based on the client's default buffer size.

For Cray systems running UNICOS , the default buffer size (default `rsize` and `wsize`) is 8 Kbytes, with a maximum of 32 Kbytes (`NFS_MAXDATA`). Most systems that are not Cray systems have both their default and maximum buffer sizes set to 8 Kbytes. Therefore, generally, NFS client systems do not need to specify `rsize` and `wsize`; instead, the default of 8 Kbytes is used. One exception to this is when NFS is being run between two Cray systems; for better performance, you should set `rsize` and `wsize` on the `mount` command to 32 Kbytes instead of using the default of 8 Kbytes.

3.4.1.3 NFS Daemons

Other factors that affect NFS performance are the number of NFS daemons (`nfsd`(8)) running on the NFS server and the number of block I/O daemons (`biod`(8)), if any, running on the NFS client. The `nfsd` daemons are used on the NFS server to respond to requests from NFS clients for access to exported file systems. The `biod` daemons are used on the NFS client to allow for applications to use asynchronous block I/O. The `biod` daemons on the UNICOS system provide write-behind capabilities on behalf of the application.

The typical number of `nfsd` daemons running on a UNICOS NFS server system is 4; the typical number of `biod` daemons running on a UNICOS NFS client system is 4. If your system, acting as an NFS server, is going to be used as a fileserver for the rest of the network, more `nfsd` daemons may be required. However, it is not clear how to determine whether you are running enough of these daemons. You must use a trial-and-error method in which you determine whether adding more daemons makes any improvements to NFS performance.

Note: Adding more `nfsd` or `biod` daemons does not improve performance of a single stream, but it does affect the overall performance of multiple users. However, you should run some `biod` daemons, because running `biod` daemons increases the performance of a single write stream. You should also note that `biod` daemons do use mbufs (see Section 2.3.2.1, page 129 , for details on setting the number of mbufs).

3.4.1.4 File System Configuration and `ldcache`(8)

One of the major limiting factors for any NFS server is its performance in accessing data (disk I/O performance). Many times this is the only limiting factor and therefore, performance never improves above the NFS server's disk I/O performance. For Cray systems acting as NFS servers, disk I/O performance is affected by whether `ldcache`(8) is configured and how it is used. If a Cray system is acting as an NFS server with a heavy NFS access load, using `ldcache` can greatly increase performance.

However, you should be aware of some factors. Ensure that using `ldcache` for NFS data access does not negatively affect local I/O performance (see the *UNICOS Configuration Administrator's Guide*, for more information on `ldcache` and configuring disks). Also, `ldcache` violates the NFS stateless protocol. For example, if the NFS server goes down, it should not affect the NFS client, except for a delay in processing until the NFS server comes back up. However, if the NFS server uses `ldcache` and the NFS server goes down, it may affect the NFS client, in that any NFS data that was changed or added by the client may not have been written to disk before the crash; the NFS client will not be aware that the changes were not made to the disk. Of course, similar concerns also affect local I/O requests using `ldcache`.

3.4.1.5 Network Speed

The network speed of channel devices such as the 10 Mbit/s Ethernet, 100 Mbit/s FDDI, 50 Mbit/s HYPERchannel, or 800 Mbit/s HIPPI affects performance. Generally, the faster the network, the better the performance. However, there can be exceptions because network speed is not the only factor that affects performance. One network can be faster than another network, but it can have higher overhead that decreases performance.

3.4.1.6 Network Configuration and Load

Performance is poor for any network application if the network is overloaded or poorly configured. You should check your networks for signs of overload (for example, a high number of collisions) by using such utilities as `netstat`(1B) to evaluate the state of your network, and `netperf`(8) and `nfsstat`(8), which display NFS/RPC statistics.

You should also try to configure your networks to reduce the number of gateway hops that are required to get from an NFS client to an NFS server.

3.4.1.7 NFS Server/client Configuration and Load

If a system will be an NFS server for several NFS clients with heavy access, it should be configured as a dedicated NFS server system and be used only minimally for other purposes. If an overloaded system is acting as either an NFS server or client, performance will be poor.

3.4.2 Obtaining NFS Performance Figures

NFS performance figures can be obtained by running the NFS performance test, `fileperf`, included in the NFS source directory, `/usr/src/net/nfs/tests/nfsperf`. Tests should be run with the Cray system acting as an NFS server and run again with the Cray system acting as an NFS client. See Section 3.3.6, page 292, for more details.

If possible, you should run the same tests through a similar network configuration, between two systems that are not Cray systems, to obtain figures that can be compared to the Cray system performance figures.

To obtain peak performance numbers for a particular configuration, follow these guidelines:

- Use the fastest networks available for NFS access (see the previous section for exceptions to this guideline).
- Ensure that the client machine, the network, and the server machine are dedicated for the test.
- Eliminate (or at least reduce) gateway hops.
- Set the value of `NFS_MAXDATA` on the NFS server and client to the optimal size. By default, Cray systems set `NFS_MAXDATA` to 32 Kbytes; it is desirable, but not always possible, to match this value on the other system. See Section 3.4.1.1, page 294, for more information on setting this value.
- Set the mtu for the interface to the optimal size (see Chapter 2, page 3, for details on setting the mtu size).
- Use `ldcache`(8) on Cray systems, or comparable caching methods on other systems, for the file systems on the NFS server.
- Run `biod`(8) daemons on the NFS client system.

NFS peak performance figures are usually no better than 50% of peak performance rates obtained with `ftp` (in some cases, NFS figures may be considerably less than 50%). One of the major factors for these statistics is that NFS is a synchronous protocol.

Network Information Service (NIS) [4]

Warning: The network information service (NIS) feature is not part of the Cray ML-Safe configuration of UNICOS. This chapter does not contain any further warnings or information pertaining to the use of the Cray ML-Safe configuration of UNICOS.

4.1 About NIS

The UNICOS network information service (NIS) facility (formerly known as yellow pages) is a network service that allows information such as passwords and group IDs for an entire network to be held in one database. Implemented along with the Remote Procedure Call (RPC) and eXternal Data Representation (XDR) library routines, UNICOS NIS has the following features:

- Look-up service. UNICOS NIS maintains a set of databases that can be queried through the use of pointers, or *keys*. Programs can request the value associated with a particular key, or all of the keys, in a database.
- Network service. Programs do not need to know the location of data or how it is stored. Instead, they use a network protocol to communicate with a database server that contains the information.
- Distributed service. Databases are fully replicated on several machines, known as *NIS servers*. The servers propagate updated databases among themselves, ensuring consistency.

NIS+ was designed to accommodate large and complex networks by allowing administrators to organize users into a hierarchical structure that reflects their interactions and not physical network setups. This structure, which is a collection of network information about users, machines, and privileges, is called the NIS+ *namespace*. You define the namespace with this information in a series of tables to arrange how your organization is best structured to fulfill and share computing needs.

The namespace is organized by domains. Each domain has a principal or *master* server and at least one backup or *replica* server. Information about each user, workstation, and domain in the namespace is organized in NIS+ tables. Twelve different NIS+ tables are referenced in an NIS+ database on a UNICOS system. NIS+ tables are maintained on both master server and replica servers, and changes to network information are easily made from any server and quickly propagated from the master server to other replicas.

The structure of the NIS+ namespace and information about its machine and human members is protected by a security system that determines the privileges of any user making requests. User access requests to network information tables can be finely controlled by NIS+ security.

The UNICOS NIS environment includes at least one Cray computer system and one or more front-end machines that are used to access the Cray system. Because UNICOS NIS differs from the NIS facility used on other systems based on the UNIX system, administration of an NIS domain that includes a Cray system must be different from administration of an NIS domain that does not.

If you are unfamiliar with NIS, it is recommended that you begin by reading the NIS documentation for the front-end system. When you have familiarized yourself with the general NIS mechanism, read the following sections to familiarize yourself with UNICOS NIS:

- NIS databases
- NIS maps
- NIS domains
- Servers and clients
- Masters and slaves
- Naming
- Data storage
- Supported databases
- Using NIS

Note: Information on network information service plus (NIS+) is presented at the end of this chapter. NIS+ software is part of SunSoft’s Open Network Computing plus (ONC+) product now shipped with the UNICOS operating system. UNICOS customers must purchase a separate license in order to use NIS+. See your license file, `/etc/craylm/license.dat`, to determine if the required license is installed.

For more information about NIS, see the Sun manual *YP Protocol Specifications.*

4.2 NIS Databases

UNICOS NIS contains network-wide databases. Usually, these databases contain files, such as `/etc/passwd` and `/etc/group`, that previously resided in directory `/etc`. However, users can add their own databases. Without the NIS service, each machine on the network would have its own identical copy of certain administrative files (for example, the `/etc/group` file on each machine would have to be updated with the same entry each time you added a user to the network).

UNICOS NIS can serve many databases. Servers containing copies of the databases are spread throughout the network. For example, when a machine in the network must look up something in `/etc/passwd`, it makes an RPC call to one of the servers to get the information. One server is the *master*; the other servers are the *slaves*. Only the database on the master server can be modified. The slaves can be periodically updated so that their information is the same as that of the master (see Section 4.6, page 303, for more information on this relationship).

4.3 NIS Maps

NIS databases contain *NIS maps*. Each map contains a set of keys and associated values. For example, the host map contains (as keys) all host names on a network and (as values) the corresponding Internet addresses. Each NIS map has a map name; programs use the map name to access data in the map.

A program must also know the format of the data in the map it wants to access. On Cray systems, the information in the NIS map is usually identical to the information in the `/etc/passwd` and `/etc/group` ASCII files. The maps are implemented in `dbm`(3C) format in the subdirectories of `/etc/yp` on NIS server machines.

4.4 NIS Domains

An *NIS domain* is a named set of NIS maps. An NIS server contains all of the NIS domain maps in a subdirectory of `/etc/yp`; this subdirectory is named after the domain. You can determine the name of your NIS domain by executing the `domainname`(1) command.

You must use a domain name to retrieve data from an NIS database. For example, if your NIS domain is `menagerie`, and you want to find the Internet address of host `dbserver`, you must use the following command to ask NIS for the value associated with the `dbserver` key in the `hosts.byname` map within the NIS domain `menagerie`:

```
ypmatch -d menagerie dbserver hosts.byname
```

Maps for the `menagerie` domain would be in subdirectory `/etc/yp/menagerie`. Each machine in the network belongs to a default domain, set when `/etc/ypstart` is entered at boot time. The `domainname` command is entered and sets the domain name to the domain that is configured in `/etc/config/ypdomain.txt`. For information on changing a domain name, see Section 4.10.3, page 308.

4.5 Servers and Clients

Servers provide resources; *clients* use them. However, neither a server nor a client is necessarily restricted to one role. For example, consider the following services:

Service	Description
UNICOS NFS	If a server machine has exported a file system, UNICOS NFS allows client machines to mount the file system remotely and to access files in place. However, a server that exports file systems can also mount remote file systems exported by other machines, thus becoming a client; therefore, a specific machine can be both server and client, client only, or server only.
NIS	The NIS server is a process that runs on a machine that might not be an NFS server. A process can request information from the NIS database, making it unnecessary to have such information on all machines. All processes that use NIS services are NIS clients. Clients can be served by NIS servers on the same machine or by NIS servers that run on another machine. If a remote machine that runs an NIS server process crashes, client processes can receive NIS services from another machine. Consequently, NIS services are usually available.

4.5.1 Servers

To become a server, a machine must have the NIS databases and run the NIS daemon `ypserv`(8). The `ypinit`(8) command automatically invokes this daemon and contains a flag that indicates whether you are creating a master or a slave server. When you update the master copy of a database, you can use the `yppush`(8) command to send the changes to the slave server. Conversely, you can use the `ypxfr`(8) command from the slave server to receive any changes from the master.

The makefile in `/etc/yp` first uses the `makedbm`(8) command to create a database, and then calls the `yppush`(8) command to send the change throughout the network.

4.5.2 Clients

A client machine that is not a server does not access local copies of `/etc` files (except for the `/etc/passwd` and `/etc/group` files); instead, it makes an RPC call to an NIS server each time it needs information from an NIS database. The `ypbind` (see `ypserv`(8)) daemon retains the name of a server. When a client boots, `ypbind` broadcasts, requesting the name of an NIS server. Similarly, if the old server crashes, `ypbind` broadcasts, requesting the name of a new NIS server. The `ypwhich`(1) command gives the name of the server to which `ypbind` currently points.

You can use the `ypcat`(1) and `ypmatch`(1) commands to read and search files because client machines do not contain entire copies of files in the NIS database. For example, to search for a user's password entry, enter one of the following commands:

```
ypcat passwd | grep username
ypmatch username passwd
```

4.6 Masters and Slaves

An NIS server is either a master or a slave. For any map, one NIS server is designated the master; all changes to the map should be made on that machine. The changes are then propagated from NIS master to NIS slaves. When a map is built by the `makedbm`(8) command, it is internally time-stamped. If you build an NIS map on a slave server, you temporarily break the NIS update algorithm and must then synchronize all versions manually.

Different maps can have different servers as the master. A given server can also be master with regard to one map, and slave with regard to another. To avoid confusion, however, it is recommended that one server be designated as master for all maps created by the `ypinit`(8) command in one domain. The examples in this chapter assume that one server is the master for all maps in a given domain.

4.7 Naming

You can use the `domainname`(1) command and the `getdomainname` library routine (see `getdomain`(3C)) to give two networks different domain names. The NIS databases for each domain name is stored in the `/etc/YP/`*domainname* directories. Thus, the `passwd.byname` map for the `menagerie` domain is stored as `/etc/yp/menagerie/passwd.byname/pag` and `/etc/yp/menagerie/passwd.byname/dir.`

For example, assume that a company has two different networks, each with its own separate list of hosts and passwords. Within each network, the user names, numerical user IDs, and host names are unique. However, some duplication occurs between the two networks. Although one domain name should be used whenever possible, in this case, the `hostname`(1) command and the `gethost`(3C) library routine would not be able to identify a host or user name uniquely. Therefore, the `domainname` command and the `getdomainname` library routine should be used to identify the domain in which a particular host or user name can be found.

4.8 Data Storage

The NIS data is stored in `dbm`(3C) format. For example, the NIS map `passwd.byname` for the `menagerie` domain is stored on an NIS server as `/etc/yp/menagerie/passwd.byname.pag` for the file that contains only data, and as `/etc/yp/menagerie/passwd.byname.dir.` for the directory that contains a bit map. The `makedbm`(8) command takes an ASCII file such as `/etc/passwd` and converts it into a `dbm` file suitable for NIS to use.

4.9 Supported Databases

Cray supports the `publickey`, `passwd`, `group`, and `netgroup` NIS files. The `publickey` and `netgroup` files function the same on Cray systems as on Sun systems. The `passwd` and `group` files function differently; they are described in the following sections.

4.9.1 The `/etc/passwd` File

The `/etc/passwd` file is stored as two separate maps in the NIS database. The first map, `passwd.byname`, is indexed by login name. The second map, `passwd.byuid`, is indexed by user ID.

You can also reference the maps by nicknames. For example, when you use the `ypcat`(1) command to display or print information about a map, and you use the map's nickname, the routine translates that nickname into the actual name of the map. Therefore, `ypcat passwd` is translated into `ypcat passwd.byname`, because no file is named `passwd` in the NIS database. The `ypcat -x` command furnishes a list of map names and nicknames.

The user database (UDB) generates the `/etc/passwd` file. The UNICOS system requires the use of the UDB for login control, accounting, and user limits. Therefore, a user must have a UDB entry in order to log in to a Cray system that runs the UNICOS system. The UDB file is searched first when a user calls the `getpwent`(3) library routine.

If password control is to be administered from the NIS database rather than from the UDB, all users should be set up in the UDB with an empty password field, and the UDB flag `permbits:YP:` must be set in the UDB entry for each user whose password is to be maintained by NIS. This method is analogous to the method used on other systems based on the UNIX system, in which each user listed in the NIS password map has the following entry in the `/etc/passwd` file:

+*user*::*uid*:*gid*:::

These entries direct NIS to fill in the missing fields from the NIS password database. (The colons are delimiters for the login name, password, uid, gid, comment, home directory, and shell fields, respectively.)

For more information on the UDB, see *General UNICOS System Administration*.

The generation of the `/etc/passwd` file is done automatically by certain UDB commands. The password file generated by the UDB is merged with the master copy of the password file that resides on the master server. The login name, user ID, and default group ID fields in the UDB-generated password file must be identical to those in the master copy when the files are merged. Because these fields are required to be present in the UDB, they will never be empty in the UDB-generated password file. However, if any of the password fields in the UDB-generated password file are empty, you must enter the associated passwords into the master copy when merging the two files.

4.9.2 The `/etc/group` File

On Cray systems, the `/etc/group` file is read and sometimes modified by the UDB. However, the group file does not depend on the UDB as much as the password file does. The UDB can modify the group file only when a numerical group ID is assigned to a UDB entry and that numerical ID does not appear in the group file. In this case, the UDB generates a new group name for that numerical ID and adds the numerical ID and the new name to the end of the group file.

The group file can be administered as on any UNIX system, even though it can be modified by the UDB commands. However, because the UDB modifies the `/etc/group` file directly, the `/etc/group` file on the Cray system should be without a default `+` entry.

4.9.3 Changing NIS Data

Each time the `/etc/passwd` file or `/etc/group` file is modified on the master NIS server, a new NIS database should be generated. You can use the `passwd`(1) command to change the password before generating the new NIS database or you can use the `yppasswd`(1) command to change the password after generating the new NIS database.

To change other data in the NIS, you must log in to the master server and edit databases there; the `ypwhich`(1) command tells you which NIS server to use.

To change their passwords in the NIS database, users of Cray systems should use the `yppasswd` command (this command works only if you have started the `yppasswdd`(8) daemon on the NIS master server machine).

4.10 Using NIS

The primary goal of NIS is to maintain one administrative environment for the machines in a local network. The NIS facility maintains a consistent set of login names and IDs across multiple machines. When you configure NIS into the UNICOS operating system, the Cray system becomes a participating member of one administrative environment. UNICOS commands and library calls include the necessary code to support access to NIS databases. Although you can use NIS to access other information, Cray currently supports the distribution of only the following databases:

- `group`
- `passwd`

- `publickey`
- `netgroup`

The following sections describe the relationship of UNICOS NFS to NIS, NIS installation procedures, NIS domain configuration procedures, the procedure for adding a user to the UNICOS NIS domain, precautions concerning procedures not recommended by Cray, and secure RPC.

4.10.1 NIS and UNICOS NFS

The UNICOS network file system (NFS) is designed to be used within one administrative environment (such as an NIS domain). Use of NIS ensures that login names and, most importantly for NFS, numerical user IDs and group IDs are unique for all users in the administrative environment.

The need for ID mapping in UNICOS NFS is reduced when the Cray system is a member of an NIS domain. Although the ID mapping process incurs minimal overhead, the ID mapping tables reside in kernel memory. However, the Cray system is typically placed in the middle of several (perhaps many) different administrative environments. Therefore, ID mapping can still be used between the Cray system and any host or network that is not part of the NIS domain of which the Cray system is a member. In this case, the UDB is generally a superset of the NIS password and group maps. See Chapter 3, page 235, for more information on UNICOS NFS and ID mapping.

4.10.2 Configuring NIS

If you are upgrading UNICOS and are using the conversion utility, the NIS feature is on or off, depending on whether the feature was turned on or off in your UNICOS configuration (prior to the upgrade). Otherwise, the NIS feature is on by default.

If you are using the UNICOS Installation and Configuration Menu System (ICMS), see the `Configure System -> Major software configuration` menu for the selection that turns on the NIS feature.

If you are not using the UNICOS ICMS for your configuration, you can turn on the NIS feature by modifying the `/etc/config/config.mh` file. Change the following line

```
#define CONFIG_YP 0
```

to read as follows:

```
#define CONFIG_YP 1
```

After you make this change, follow the remaining system build procedures outlined in the *UNICOS System Configuration Using ICMS*.

4.10.3 UNICOS NIS Domain Configuration Procedure

After NIS has been installed, you are ready to configure the Cray system as a slave server. Making the Cray system a slave server instead of a master server or a client provides the most efficient performance.

It is not recommended that you configure the Cray system as a master server, because responding to requests from other clients that are binding to the Cray server uses Cray cycles; this is not the most efficient use of the capabilities of the Cray system. Also, as a master server, the Cray system can overload the network with NIS traffic.

It is not recommended that you configure the Cray system as a client because the speed of the system overloads the NIS server, and because the performance of user programs is slowed when the network is accessed for look-up functions.

Use the following steps to configure the Cray system as a slave server:

1. Before initializing the system for multiuser mode, set the NIS domain name to null, as follows:

   ```
   domainname ""
   ```

2. Initialize multiuser mode, or initialize and configure the network.

3. Specify the domain name on the Cray system to be the same as the name of the domain for which it is to be a slave server, as follows:

   `domainname` *domain*

4. Run the `ypinit`(8) script (this needs to be done only once), as follows:

   `/etc/yp/ypinit -s` *YP_master_server_hostname*

5. Reset the domain name to null, as follows:

   ```
   domainname ""
   ```

   This prevents error messages from occurring during any unintentional reference to NIS until the remaining configuration is complete.

The Cray system is now known to the NIS master server as a slave server; that is, the host name of the Cray system has been added to the NIS database `ypservers`. The Cray system now has copies of the NIS databases that it supports.

If you are using the startup procedures provided with the UNICOS system (that is, the `/etc/ypstart` file), and if you turned on the NIS feature during installation, you can specify the NIS domain name by placing the name in the `/etc/config/ypdomain.txt` file. One way of specifying the NIS domain name in this file is as follows:

```
echo your_NIS_domain_name > /etc/config/ypdomain.txt
```

The `/etc/ypstart` script accesses this file to set the NIS domain name and then starts the required NIS daemons.

If you are using the UNICOS ICMS for your configuration, use the `Configure System -> Network configuration -> NIS configuration` menu to set the contents of this file.

If you are not using the startup procedures provided with the UNICOS system (that is, if you are using a modified `/etc/netstart` file or another script of your own creation), add the following commands to your start-up file:

```
domainname your_NIS_domain_name
ypserv
ypbind -h ‘hostname‘
```

If you want to use secure RPC, you must start up the `keyserv` process by invoking the `/etc/keyserv` daemon in the `/etc/netstart` script.

4.10.4 Adding a User to the UNICOS NIS Domain

If the NIS domain has been configured as recommended in Section 4.10.3, page 308, you can add a user to the NIS domain, as follows:

1. Choose a user ID for the new user. Examine the UDB and the NIS database to ensure that the user ID is unique.
2. Add the user to the UDB. Ensure that the `NIS_PERMBITS` flag is set in the `PERMBITS` field of the UDB entry (this flag indicates the use of UNICOS NIS). If password control is to be administered from the NIS database rather than from the UDB, you must leave the password field empty.
3. Copy the new user’s UDB-generated `/etc/passwd` file entry into the `/etc/passwd` file that resides on the master server.

4. As `root`, run `passwd`(1) on the NIS master server machine to give the new user a password, because the UDB-generated password file entry does not include one.

5. Remake the NIS database.

6. Create the new user's home directory. You must determine whether the home directory on the Cray system is the same as that in the rest of the NIS domain.

   Note: When a user is added to or deleted from the UNICOS NIS domain, both the UDB and the NIS master server's password database reflect the change.

4.10.5 Precautions Concerning Sets of Users

It is possible, but not recommended, to have an intersecting set of users in the NIS database and the UDB; that is, systems other than Cray systems might have users who are allowed to use the Cray system and users who are not allowed to use the Cray system. Consequently, some users on a given system would be listed in the UDB, and some would not. Although users listed in the NIS databases can log in anywhere else within the NIS domain, only users listed in the UDB can use the Cray system. Such an environment defeats the purpose of putting the Cray system into an NIS domain, unless this domain is one of many administrative environments that include the Cray system.

When intersecting sets of users are present, you must be careful when merging the UDB-generated password file with the password file that is used to generate the NIS database. Take the following precautions:

- Ensure that users who are allowed to use the Cray system and are members of the NIS domain have identical login names, user IDs, and group IDs in the UDB and the NIS databases.
- Ensure that users who are allowed to use the Cray system but are not members of the NIS domain do not have their UDB login names, user IDs, or group IDs in the NIS database.
- Ensure that users who are not allowed to use the Cray system but are members of the NIS domain do not have their NIS database login names, user IDs, or group IDs in the UDB.

4.10.6 Precautions Concerning the Cray System As a Master Server

If the Cray system is configured as a master server, it is recommended that no other machines bind to it.

You cannot use the `/etc/passwd` and `/etc/group` files to generate the NIS password and group maps. Copies of these files must be placed elsewhere; the information missing from the UDB must be added manually. This makes password control more difficult.

You cannot use a standard command to give a user a password before making the NIS password map. The `passwd`(1) command modifies the UDB (not the `/etc/passwd` file). The `yppasswd`(1) command modifies the NIS password map. However, you should assign the user a password before building the NIS database. Therefore, you must use the `-f` option of the `passwd`(1) command to change the password field in a `passwd`(5) format file, rather than in the UDB.

4.10.7 Precautions Concerning NIS and UNICOS Security

If UNICOS security features are enabled on your Cray machine, you must take special precautions to ensure that NIS will operate properly. Specifically, all local host network interfaces must be added to the network access list (NAL) in the network security file `/etc/config/spnet.conf`. For more details, see the `spnet`(8) command. As a result, NIS processes (such as `portmap`, `ypserv`, and `ypbind`) can communicate with each other through all local interfaces.

4.10.8 Secure RPC

The secure Remote Procedure Call (RPC) subsystem is the means by which the `AUTH_DES` style of RPC authentication is implemented. See the *Remote Procedure Call (RPC) Reference Manual*, for details of RPC authentication.

Each user of secure RPC must have an entry in a special NIS database of public and private keys. There must be one such entry for each host that the user accesses. Similarly, each host that supports secure RPC must have an entry for each server it accesses.

The NIS database file is called `publickey.byname` and it consists of 16 characters. This implies that, if secure RPC is to be run, this file must exist on a file system that supports names of this length.

The following sections describe methods for generating the database and developing applications.

4.10.8.1 Generating the Database

Issue the `newkey`(8) command to add entries to the `/etc/publickey` file. This command creates public key / private key pairs for users and hosts on the network. A *public key* is accessible to all users; a *private key* is encrypted according to the Data Encryption Standard (DES) with the existing password, and it is accessible only to the user or host to which it is assigned.

To add a user to the database, enter the following command:

```
/etc/newkey -u username
```

The `newkey` program prompts for the login password of the user specified by *username* and then creates a unique public key / private key pair for that user.

To add a host to the database, enter the following command:

```
/etc/newkey -h hostname
```

The `newkey` program prompts for the root password of the host specified by *hostname* and then creates a unique public key / private key pair for that host.

4.10.8.1.1 Database Format

Secure RPC authentication uses a cryptographic scheme that allows each client / server pair to obtain a unique key with which authentication data can be encrypted. The entries in the `publickey.byname` database are of the following format:

opsys.id@domain publickey:*privatekey*

The fields of the entry are as follows:

opsys	Operating system. This field is always `unix` in the current implementation.
id	UNICOS or UNIX identifier for the user or host. For users, it is the same as the *uid* field in the `passwd` file. For hosts, it is the full name of the host, which can be obtained by executing the `ypwhich`(1) command.
domain	Name of the NIS domain within which the database resides.
publickey	String of 48 hexadecimal digits that represent a 192-bit public key for this ID. This public key is

	stored in a nonencrypted form and can be read by any user on the system.
privatekey	String of 48 hexadecimal digits that represent a 192-bit private key, corresponding to the previous public key field. Unlike the public key field, the private key field is stored in a DES-encrypted format. Only the user or host specified by *id* can obtain access to the private key in its nonencrypted format.

After the new users and hosts are added to the `/etc/publickey` file, you must rebuild the `publickey.byname` database. If the master server is a Cray machine (which Cray does not recommend), enter the following command in the `/etc/yp` directory:

```
make -f yp.mk publickey
```

Note: The secure RPC subsystem depends on the existence of the NIS database file of public and private keys. This implies that NIS must be configured on the Cray system if it is to support secure RPC.

4.10.8.1.2 Database Access

The `keyserv`(8) program accesses the `publickey.byname` database, encrypts and decrypts private keys, and performs the relatively complex mathematics used to implement the public key system used by secure RPC. The `keyserv` process is a daemon that is usually started up from the `/etc/netstart` (or equivalent) script.

A `keyserv` process must run on both the client and the server machines. This process can be run only by `root`, and it binds to a reserved port on a user datagram protocol (UDP) socket. `keyserv` registers itself with `portmap`, and it is always program number 100029.

To determine whether a `keyserv` process is active on any given host, users can enter the following command:

```
rpcinfo -u hostname 100029
```

The `rpcinfo` command either returns the following error message:

```
rpcinfo: RPC: Program not registered
program 100029 is not available
```

or returns the following completion message:

```
program 100029 version 1 ready and waiting
```

Communication between a client or server and the `keyserv` process is accomplished through a process called `keyenvoy`(8). The `keyenvoy` process is `setuid root` and cannot be run interactively. It is created and destroyed dynamically by the RPC library routines in `libc`. The `keyenvoy` process creates a secure communications channel between a client or server process and `keyserv`. The `keyenvoy` process communicates with `keyserv` through secured local RPC channels. `keyenvoy` communicates with the client or server process through `stdin` and `stdout`. This process should be completely transparent to all users.

The `keylogin`(1) program informs `keyserv` that a user is interested in using the secure RPC subsystem. `keyserv` then caches the public key / private key entry for this user so that it does not have to look up this information at run time.

Usually, there is no reason for the user to run the `keylogin` program, because the `login`(1) program informs `keyserv` of new users. However, if for some reason the `login` program fails to communicate with `keyserv`, or if the `keyserv` program crashes and must be restarted, or if a user has a null password, the `keylogin` program can be invoked to inform `keyserv` directly of the user's interest in using secure RPC.

When `keylogin` is invoked, the user is prompted for the password. If the password is incorrect, this error is reported to the user. Unless an error message indicates otherwise, the password has been accepted by the RPC subsystem, and `keyserv` has cached the relevant key information successfully.

4.10.8.2 Developing Secure RPC Applications

To prevent false identification, some restrictions must be observed in developing secure RPC applications.

If a client wants to use a server process that is running as `root`, the client must specify the network name of the host as the first parameter of the `authdes_create()` call. This specification ensures that the server that is running on the remote host is actually a `root` process. The network name of the host can be obtained by making a call to the `host2netname` routine, as follows:

```
char    netname[MAXNETNAMELEN+1];
char    *hostname;
char    *domainname;
hostname = "mycray";
domainname = "my_yp_domain";
host2netname(netname, hostname, domainname);
```

When this call returns, the network name of the host is written in array `netname`. If the `domainname` pointer is null, the `host2netname` routine uses the default domain name.

If a client wants to use a server process that is not running as `root`, the user must know the network name of the user who is running it. This name can be obtained through a call to the `user2netname` routine, as follows:

```
char    netname[MAXNETNAMELEN+1];
int     uid;
char    *domainname;
uid = 134;      /* assume remote user has uid 134 */
domainname = "my_yp_domain";
user2netname(netname, uid, domainname);
```

When this call returns, the network name of the user is written into array `netname`. Again, if the domain name pointer is null, the `user2netname` routine uses the default domain name. This network name can then be passed as the first argument to the `authdes_create` call.

4.11 About NIS+

The following sections briefly describe the administration basics of the network information service plus (NIS+) facility. This information is aimed at system and network administrators who have a working knowledge of NIS version 2 and client-server networks. If you are unfamiliar with NIS, you should first read the previous sections in this chapter.

For an introduction to NIS+ that describes conceptual aspects and pre-setup planning strategies, see *ONC+ Technology for the UNICOS Operating System*. This guide contains a discussion of enterprise networks, NIS+ domains, NIS+ objects, and the hierarchical structure of the NIS+ namespace. The following sections represent a subset of information on the functionality of NIS+. See also Rick Ramsey’s book, *All About Administering NIS+*, for more detailed information on NIS+ operations, and migration from NIS to NIS+.

The following sections briefly describe what NIS+ is and how to set it up on your UNICOS system:

- NIS+ licensing
- NIS+ overview
- Comparing NIS and NIS+

- Components of NIS+
- Planning your NIS+ namespace
- Setting up your first NIS+ domain
- How to set up a root domain
- Initializing an NIS+ client
- Setting up an NIS+ server
- How to set up a nonroot domain
- Administering your NIS+ namespace
- Migrating from NIS to NIS+

4.12 NIS+ Licensing

Network Information Service plus (NIS+), is a trademarked software product developed for the Solaris operating system by SunSoft Inc., a Sun Microsystem company. NIS+ is one of several software products that make up SunSoft’s Open Network Computing plus (ONC+) product line.

Under agreement with SunSoft, Cray has ported NIS+ for the UNICOS operating system. Although this NIS+ software is packaged with the current release of UNICOS, it is a separately licensed product. You must purchase a Cray license to use NIS+ and other SunSoft ONC+ products on your UNICOS systems.

To see if you already have a Cray license for NIS+, enter any of the NIS+ commands, such as the following `nisls`(8) command:

```
nisls
```

If you do not have an ONC+ license, the following error message appears on your screen:

```
ONC+ license required
```

If you would like to purchase a Cray license to use NIS+ or other ONC+ software, contact your purchasing department or your Cray sales representative. NIS+ is a new network information service that was created to replace NIS.

4.13 Comparing NIS and NIS+

As computer networks become larger and more complex, administrative complexity grows as well. The model of central administration for computing networks is obsolete because keeping track of information for all the users and machines in a enterprise network is labor intensive and slow to update.

NIS+ has many advantages over NIS. Speed and security are the most prominent differences. A comparison of NIS and NIS+ is summarized in the following table.

Table 6. Comparing the features of NIS and NIS+

Feature	NIS	NIS+
Organization	Flat domains	Hierarchical structure
Data storage	Bicolumn maps	Multicolumn tables
Security	No authentication available	DES authentication
Information	One network service	Multiple network services (NIS, NIS+, DNS, or local files)
Server updates	Delayed batch propagation	Immediate incremental updates

4.14 Components of NIS+

NIS+ consists of the following main components:

- NIS+ namespace
- Directory objects
- NIS+ domains
- NIS+ servers
- NIS+ clients
- NIS+ tables
- Name service switch

4.14.1 NIS+ Namespace

The NIS+ namespace is a collection of network information stored by NIS+. The namespace can be configured in a variety of ways to fit the needs of an organization. Although the structure of an NIS+ namespace can vary from site to site, all sites use the same structural components including directories, tables, and groups. These components are called NIS+ objects. There is always only one namespace per computing environment.

4.14.2 Directory Objects

Directory objects define the sections of the namespace. When these directory objects are arranged in a tree-like structure, they divide the namespace into separate parts. A namespace can have several levels of directories. The topmost directory is called the *root* directory. A namespace with only one directory is defined as *flat*.

With two or more directories arranged in a hierarchical organization, an NIS+ namespace looks similar to the way UNIX directories are organized in hierarchical file systems. UNIX directories, however, are designed to hold files, and NIS+ directories are designed to hold NIS+ objects.

NIS+ objects include other directories, tables and groups containing information about the network's machines, processes and users. Any NIS+ directory that stores NIS+ groups is named `group_dir`. Any directory that stores NIS+ system tables is named `org_dir`. NIS+ system tables contain several categories of network information and they are briefly described in this section.

4.14.3 NIS+ Domains

Domains are not tangible locations or physical objects of the NIS+ namespace. They are names for sections of the namespace and therefore they reflect the organization of users and machines in a computing environment. Domains are designated to support separate portions of the namespace. An NIS+ domain consists of a directory object, its `org_dir` directory, its `groups_dir` directory, and a set of NIS+ tables. Each domain contains client machines and servers to support the users working in that section of the namespace. The servers keep track of who (clients) is on the network and what access rights they have.

4.14.4 NIS+ Servers

The servers store the domain's directories, groups, and tables. They answer requests for access from users, administrators, and applications. Each domain is supported by one set of servers, but that same set of servers can also support more than one domain. A server that supports a domain is not associated with the domain but with the domain's directory.

When the connection between the server and the directory is established, the directory stores the name and Internet Protocol (IP) address of its server. This information is used by clients to send requests for service.

Two type of servers support an NIS+ domain: *master* and *replica* servers. The master server of the root domain is called the *root master server*. A namespace has only one root master server. Both master and replica servers store NIS+ tables and answer client requests. The master server stores the master domain tables. The replicas store only duplicates. The administrator loads information into the tables in the master server, and the master server propagates the information to replica servers. Any workstation can be set up as either a master server or a replica server. Sometimes a master server is a workstation that has more disk capacity to handle the many client requests.

4.14.5 NIS+ Clients

An NIS+ client is a workstation that has been set up to receive NIS+ service. An NIS+ client can access any part of the namespace subject to security constraints. If the client has been authenticated and if it has been granted permission rights, the client can access information or objects in any domain in the namespace. When a client requests access to the namespace, it is actually requesting access to a particular domain in that namespace. Therefore, a client sends its request to the server that supports the domain it is trying to access.

4.14.6 NIS+ Tables

There are at least 11 standard NIS+ tables that contain network information about your namespace. NIS+ tables on UNICOS systems are listed in Table 7, page 320. These tables have a column-entry structure and, therefore, have several advantages over other sources of network information, such as flat bicolumn maps or local flat `/etc` files. All NIS+ tables have the same structure of rows and columns. A client with proper access rights can access the information of the whole table or by an individual key, column or entry. NIS+ software examines the privileges of a user by *authentication*, and either allows or refuses access requests to network information by a process called *authorization*.

Custom NIS+ tables can be set up by administrators. They can be searched individually by entry or colum, or several can be symbolically linked or connected by a concatenation path.

Table 7. NIS+ tables on UNICOS systems

NIS+ table	Fields	Table information for each domain
`auto_home`	Mount point Options and locations	Names of users and the locations of their home directories
`auto_master`	Mount point Map name	Mount points and names of automounter maps
`cred`	Secret key Public key	Credentials of client workstations and client users
`group`	Name Password GID Numbers	Names, passwords, IDs, and members of UNICOS groups
`ethers`	Ethernet address Official host name	Names and Ethernet addresses of workstations
`netgroup`	Group name List of members	Names of network groups and lists of their members.
`networks`	Network name Network number Aliases	Internet names, and numbers of networks and their aliases
`protocols`	Protocol name Protocol number Aliases Comments	Names, numbers, and aliases of protocols used by the Internet

NIS+ table	Fields	Table information for each domain
`rpc`	RPC program name RPC program number Aliases Comments	Names, numbers, and aliases of RPC programs
`services`	Service name Port / Protocol Aliases Comments	Names, port numbers, and aliases of Internet services
`timezone`	Time zone name Workstation / Domain name Comments	The default time zone and names of workstations or one domain name

4.14.7 Name Service Switch

NIS+ interacts in a different way with `/etc` files than NIS. NIS+ interacts with all network information services, including `/etc` files, through the *name service switch.* A configurations file is located on every NIS+ client and lists the sources for network information for that client. Sources of network information include:

- `/etc` files
- NIS bicolumn maps
- Domain name service (DNS) zone files
- NIS+ tables

NIS+ lookups follow the look up sequence defined by the configuration file on NIS+ clients. A sample configuration file is set up like the following table.

Table 8. Sample NIS+ client configuration

Type of information	Service	Look up sequence
`passwd:`	`files`	`nisplus`
`group:`	`files`	`nisplus`
`hosts:`	`nisplus`	`[NOTFOUND=return] files`
`services:`	`nisplus`	`[NOTFOUND=return] files`
`networks:`	`nisplus`	`[NOTFOUND=return] files`
`protocols:`	`nisplus`	`[NOTFOUND=return] files`
`rpc:`	`nisplus`	`[NOTFOUND=return] files`
`ethers:`	`nisplus`	`[NOTFOUND=return] files`
`publickey:`	`nisplus`	
`netgroup:`	`nisplus`	
`automount:`	`files`	`nisplus`
`aliases:`	`files`	`nisplus`

See the `nsswitch`(4) man page for more detailed information on the name service switch and corresponding data files.

4.14.8 NIS+ Commands

The following NIS+ administration commands control the setup and maintenance of the NIS+ namespace. The printed man pages for these commands are found in Chapter 8 of the *UNICOS Administrator Commands Reference Manual*. Man pages describe in detail the options and parameters of command use.

Table 9. NIS+ administration commands

Command	Description
`nisaddcred`	Creates security credentials for NIS+ principals
`nisaddent`	Creates NIS+ tables from corresponding `/etc` files and NIS maps
`nis_cachemgr`	Maintains a cache of location information about NIS+ servers
`niscat`	Displays the contents of NIS+ tables and objects
`nischgrp`	Changes the group owner of NIS+ objects or entries

Command	Description
`nischmod`	Changes the access rights of NIS+ objects or entries
`nischown`	Changes owner of NIS+ object
`nischttl`	Changes the time-to-live value of an NIS+ object or entry
`nisctl`	Controls the operation of NIS+ servers; enters cache flushing, debugging and report printing
`nisd`	ONC RPC daemon that implements NIS+ service
`nisdefaults`	Displays NIS+ default values returned by NIS+ local name functions
`niserror`	Displays and translates NIS+ error number messages into text
`nisgrpadm`	Creates or destroys NIS+ groups and administers principals in those groups.
`nisinit`	Initializes a machine to be an NIS+ client and server
`nisln`	Symbolically links NIS+ objects
`nislog`	Displays the contents of NIS+ server transaction log
`nisls`	Lists the contents of an NIS+ directory
`nismatch`	Searches NIS+ tables by matching text strings
`nisgrep`	Searches NIS+ tables by text string or *keypat* expressions
`nismkdir`	Creates a nonroot NIS+ directory or adds a replica to an existing directory
`nispasswd`	Changes NIS+ password information
`nispath`	Prints out search path of a given NIS+ name
`nisping`	Sends a ping to all replica servers of an NIS+ directory
`nisrm`	Removes NIS+ objects from the namespace
`nisrmdir`	Removes existing NIS+ directories
`nissetup`	Initializes NIS+ domain
`nisshowcache`	Prints out the contents of the shared cache file
`nisstat`	Reports NIS+ server statistics
`nistbladm`	Creates, deletes, adds, modifies and removes NIS+ tables
`nistest`	Returns the state of the NIS+ namespace
`nisupdkeys`	Updates the public keys in an NIS+ directory object

4.14.9 NIS+ API

The NIS+ application programming interface contains 54 functions that can be called by an application to maintain and manipulate NIS+ objects. For reference, these functions can be organized into nine families. Each function name listed below is also the name of the NIS+ man page that describes its family of routines.

Table 10. NIS+ API functions

Functions	Family
`nis_subr`	Application subroutine functions
`nis_db`	Database access
`nis_error`	Error message display functions
`nis_groups`	Group manipulation functions
`nis_local_names`	Local name functions
`nis_names`	Object manipulation functions
`nis_tables`	Table access functions
`nis_admin`	Transaction log functions

Man pages describing NIS+ functions and library routines are found in an appendix in *ONC+ Technology for the UNICOS Operating System*. An API NIS+ programming example using these routines also presented in *ONC+ Technology for the UNICOS Operating System*.

4.15 Planning Your NIS+ Namespace

Before setting up NIS+ on your UNICOS system, it is recommended that you do the following tasks:

1. Diagram the structure of your organization.
2. Divide this structure into administration groups.
3. Select servers that will support each group.
4. Determine the access rights of users and groups in your organization.

Note: If an NIS namespace is already defined at your site, you can use its flat domain structure for your new NIS+ namespace. NIS+ contains scripts that allow administrators to construct an NIS+ namespace from NIS bicolumn maps or local files.

When designing your namespace and setting up your domains, consider the following factors in your plan:

- Locations of NIS+ tables

  Consider that the lower in the domain hierarchy that you store tables, the easier it is to administrate them.

- Custom NIS+ tables

  Decide which NIS maps can be converted to NIS+ tables and whether any of your current applications depend on existing NIS maps.

- Connections between NIS+ tables

  Decide whether to connect your tables by paths or links. This decision weighs convenience and performance.

  Establishing a path from tables low in one hierarchical domain to a single table in a remote domain may save look-up time because a search examines the remote table directly, without scanning the tables above it in the home domain. Another advantage to this strategy is that administrative updates to the remote domain table are visible to users across domains. Linking NIS+ tables eliminates some performance concerns because NIS+ lookups do not search a local table in the users path first, but enter a search directly on the linked table.

  Note: Although decreasing the number of tables searched increases the performance of NIS+, look-up searches across domains makes users in one domain dependent on the network availability of another domain.

The following guidelines may help you decide how to set up, customize, locate, or link NIS+ tables.

- Every domain must have access to every standard table.
- Frequently accessed data should be located as close to users in a domain as possible.
- The lower in the hierarchy data is located, the easier it is to administer.
- Data that is frequently accessed by many domains should be located higher in the hierarchy.

- NIS clients cannot access NIS+ tables that are symbolically linked or referenced by paths.

4.16 Setting up Your First NIS+ Domain

After you have diagrammed the organization of users and workstations at your site, you must consider the traffic load of your proposed NIS+ namespace. Ideally, each domain in your namespace will have one master server and a number of replicas. To make each domain efficient, it is recommended that you limit the number of replicas to two. Since each domain may have a different traffic load, you need to calculate the disk space requirements of each master server in your namespace. This calculation is described in the following section.

4.16.1 Calculating Disk Requirements for Your Master Servers

Consider the following factors when determining disk requirements for master servers:

- Disk space for `/etc/nis` (and `/etc/yp`)
- Amount of memory

NIS+ tables, objects, and groups and client information are stored in `/etc/nis`. Typically, `/etc/nis` uses about 5 Kbytes of disk space per client. An NIS+ namespace with 1000 clients requires approximately 5 Mbytes of disk space. It is recommended however, that you add an additional 10 to 15 Mbytes of disk space to handle transaction logs. It is also recommended that you checkpoint transaction logs regularly to reduce their size.

The amount of memory that is required for NIS+ servers is also determined by the size of your NIS+ domains. The minimum requirement is 32 Mbytes. We recommend that you have up to 64 Mbytes for servers that support large domains.

4.17 How to Set up a Root Domain

To set up an NIS+ root domain on your UNICOS system, perform the following steps:

1. Log into the root master server machine as `root`.

2. Set the domain name of the root master server by using the `domainname` command. For example, if you want to name your first domain `nis.com`, enter the following command line:

   ```
   domainname nis.com
   ```

3. Make sure that the `/etc/nsswitch.conf` file is configured for NIS+ service. See the `nsswitch`(4) man page for more information on the syntax of the `nsswitch.conf` file.

4. Remove any NIS+ files and kill any existing processes by entering the following command lines:

   ```
   rm -rf /etc/nis/*
   ps -aef | grep nis_cachemgr
   kill -9 $PID
   ```

5. Set the administration group for the root domain by exporting the environment variable `NIS_GROUP` for either the standard shell, Korn shell, or C shell environments by entering one of the following command lines:

   ```
   export NIS_GROUP=admin.nis.com
   ```

   (for standard or Korn shell)

   ```
   setenv NIS_GROUP admin.nis.com.
   ```

   (for C shell)

6. Initialize the root master server by entering the `nisinit` command as follows:

   ```
   nisinit -r
   ```

7. Terminate any running NIS+ daemons by searching for `nisd` processes and killing their process IDs by using the following command line:

   ```
   ps -aef | grep nisd
   kill -9 $PID
   ```

8. Start the NIS+ daemon by entering the `nisd` command with the `-r` and `-S` options as follows:

   ```
   nisd -r -S 0
   ```

   If you are setting up to run in NIS-compatibility mode, enter the `nisd` command with the `-r`, `-Y`, and `-S` options as follows:

   ```
   nisd -r -Y -S 0
   ```

9. Create NIS+ subdirectories and tables by entering the `nissetup` command as follows:

   ```
   nissetup
   ```

   If you are setting up to run in NIS-compatibility mode, enter the `nissetup` command with the `-Y` option as follows:

   ```
   nissetup -Y
   ```

10. Create the DES credentials by entering the `nisaddcred` command with the `DES` argument as follows:

    ```
    nisaddcred DES
    ```

11. Update the public key for the parent and subdirectories by entering the `nisupdkeys` command as follows:

    ```
    nisupdkeys nis.com.
    nisupdkeys org_dir.nis.com.
    nisupdkeys groups_dir.nis.com.
    ```

    See the `nisupdkeys`(8) man page for more information.

12. Search for and eliminate any NIS+ daemon that is currently running by entering the following command lines:

    ```
    ps -aef | grep nisd
    kill -9 $PID
    ```

13. Restart the `nisd` daemon with security level 2 (the default) by entering the following command:

    ```
    nisd -r
    ```

    If you are setting up to run in NIS-compatibility mode, enter the `nisd` command with the `-r` and `-Y` options as follows:

    ```
    nisd -r -Y
    ```

14. Add root's DES credentials to the root domain by entering the `nisaddcred`(8) command with the `-p` option as follows:

    ```
    nisaddcred -p unix.0@nis.com -P root.nis.com. DES
    ```

4.18 Initializing an NIS+ Client

Initialize your NIS+ clients by using the following steps:

1. Log in to the domain's master server.
2. Create DES credentials for the new client machine by entering the `nisaddcred` command with the `-p` option as follows:

   ```
   nisaddcred -p unix.mach_name@nis.com -P mach_name.nis.com. DES
   ```

3. Log in to the client machine as `root`.
4. Assign the domain by entering the `domainname` command as follows:

   ```
   domainname nis.com
   ```

5. Check that the `/etc/nsswitch.conf` file is configured for NIS+ service.
6. Remove any existing NIS+ files in `/etc/nis` and search for and kill any active processes by entering the following command lines:

   ```
   rm -rf /etc/nis/*
   ps -aef | grep nis_cachemgr
   kill -9 $PID
   ```

7. Initialize the NIS+ client by entering the `nisinit` command with the broadcast, host name, or coldstart file option. Choose and enter only one of the following command lines:

   ```
   nisinit -c -B
   ```

   (for broadcast)

   ```
   nisinit -c -H master_server
   ```

   (for host name)

   ```
   nisinit -c -C /tmp/NIS_COLD_START
   ```

   (for coldstart file)

8. Search for and eliminate any existing active processes and restart the `keyserv` daemon by entering the following command lines:

   ```
   ps -aef | grep keyserv
   rm -f /etc/.rootkey
   keyserv
   ```

9. Enter the `keylogin` command with the `-r` option as follows:

```
keylogin -r
```

10. Reboot the client.

4.19 Setting up an NIS+ Server

Set up an NIS+ server on your UNICOS system by performing the following steps.

> **Note:** Before a machine is set up as a replica, a server must be set up as the root domain, and the potential replica machine must be initialized as an NIS+ client.

1. Log in as `root` to the machine that you want to be a replica server.
2. Start the NIS+ daemon by entering the `nisd`(8) command as follows:

   ```
   nisd
   ```

   If you are setting up to run in NIS-compatibility mode, enter the `nisd` command with the `-Y` option as follows:

   ```
   nisd -Y
   ```

3. Start the NIS+ cache manager by entering the `nis_cachemgr`(8) command, as follows:

   ```
   nis_cachemgr
   ```

4.20 How to Set up a Nonroot Domain

Set up a nonroot domain on your UNICOS system by performing the following steps:

1. Log in to the domain's master server.
2. Set the administration group for the domain by exporting the environment variable `NIS_GROUP` for either standard shell, Korn shell, or C shell environments by entering one of the following command lines:

   ```
   export NIS_GROUP=admin.sales.nis.com.
   ```

   (for standard or Korn shell)

   ```
   setenv NIS_GROUP admin.sales.nis.com.
   ```

(for C shell)

3. Create the new domain’s directory and specify its servers by entering the `nismkdir`(8) command with the `-m` and `-s` options as follows:

   ```
   nismkdir -m master_server -s replica_server sales.nis.com.
   ```

   See the `nismkdir`(8) man page for information about command options and arguments.

4. Create the subdirectories and tables by entering the `nissetup` command as follows:

   ```
   nissetup sales.nis.com.
   ```

   If you are setting up to run in NIS-compatibility mode, enter the `nissetup` command with the `-Y` option as follows:

   ```
   nissetup -Y sales.nis.com.
   ```

5. Create the domain’s administration group by entering the `nisgrpadm` command with the `-c` option as follows:

   ```
   nisgrpadm -c admin.sales.nis.com.
   ```

6. Assign group members access rights to the new directory by entering the `nischmod`(8) command with the *rights* argument as follows:

   ```
   nischmod g+rmcd sales.nis.com.
   ```

   See the `nischmod`(8) man page for details about the content and syntax of the *rights* argument.

7. Add master and replica servers to the domain’s administration group by entering the `nisgrpadm` command with the `-a` option as follows:

   ```
   nisgrpadm -a admin.sales.nis.com. master_server.nis.com.
            replica_server.nis.com.
   ```

4.21 Administering Your NIS+ Namespace

Table 11 shows NIS+ commands.

Table 11. Common NIS+ commands

Task	Command line
For NIS+ groups:	
Create a group	`nisgrpadm -a` *groupname.domainname*
List members of a group	`nisgrpadm -l` *groupname*
Delete a group	`nisgrpadm -a` *groupname*
Add members to a group	`nisgrpadm - a` *groupname members* `...`
For NIS+ clients:	
Initialize an NIS+ client	`nisinit -c -H` *hostname*
Initialize an NIS+ client	`nisinit -c -B`
Initialize an NIS+ client	`nisinit -c -C` *coldstart filename*
Display all information about a workstation	`nisdefaults`
For NIS+ servers:	
Initialize the root master server	`nisinit -r`
Start a cache manager	`nis_cachemgr`
Checkpoint a database	`nisping -C` *domainname* `.com`
Display the contents of the transaction log	`nislog`
For NIS+ tables:	
Display the contents of a table	`niscat -h -A` *tablename*
Create a table	`nistbladm -c` *table-type column-spec... tablename*
Delete a table	`nistbladm -d` *tablename*
Add an entry to a table	`nistbladm -a` *entry*

Task	Command line
Modify a table entry	`nistbladm -m` *new-entry old-entry*
Remove a table entry	`nistbladm -r` *indexedname*
Load information into tables from text files	`nisaddent -t` *filename table type*
Load information into tables from NIS maps	`nisaddent -t` *NISdomain table type*
Display the object properties of a table	`niscat -o` *tablename* `.org_dir`
Search for regular expressions in NIS+ tables	`nisgrep` *expression tablename*
Search for strings in NIS+ tables	`nismatch` *string tablename*
Change the time to live value of table entries	`nischttl` [*column* = *value*], *tablename*
For NIS+ objects:	
Expand an existing NIS+ directory into a domain	`nissetup` *directory*
Create a link between NIS+ objects	`nisln` *source target*
Display the contents of a client's directory cache	`nisshowcache`
Display the time-to-live value of an object	`nisdefaults -t`
Change the time-to-live value of NIS+ objects	`nischttl` *time-to-live object name*
For NIS+ security:	
Set the security level of service	`nisd -S` *level*
Add credentials for a new user	`nisaddcred -p` [`-P`]
Remove credentials	`nisaddcred -r` *principal name*
Update or change your credentials	`nisaddcred` *credential-type*
Add access rights for an object	`nischmod` *category +right object name*
Add access rights for an entry	`nischmod` *category +right* [*column-name* = *value*] *,tablename*
Display all local domain password information	`nispasswd -a`
Display password information for a specific user	`nispasswd -d` *username*

Task	Command line
Clear public keys stored by a directory	`nisupdkeys -C` *directory*
Update the IP addresses of a server	`nisupdkeys -a -H` *server-name*

4.22 Migrating from NIS to NIS+

NIS+ can handle requests from both NIS version 2 and NIS+ clients. NIS+ has an NIS built-in compatibility mode that provides two service interfaces to answer NIS and NIS+ requests transparently. Although there is no additional setup required to run an NIS+ server in NIS-compatibility mode, the instructions for setting up in this mode are different than setting up a standard NIS+ server. An NIS server running in NIS-compatibility mode does not support the `ypupdate` and `ypxfr` protocols and, therefore, cannot be used as a master or slave NIS server.

Note: NIS servers do not provide any of the user or request credentials that NIS+ servers expect from their clients. Therefore, all NIS+ tables must have access rights set to allow unauthenticated requests in order for NIS client requests for information to be accepted and filled by NIS+ servers.

4.22.1 NIS-compatibility Mode

NIS+ uses fewer tables than NIS. Determine the information sources that you need to load by examining the mapping of local files, NIS maps, and NIS+ tables in the following table.

Table 12. Correspondence between information sources on a UNICOS system

Local files	NIS maps	NIS+ tables
`auto.home`		`auto_home`
`auto.master`		`auto_master`
`/etc/ethers`		`ethers`
`/etc/group`	`group.bygid`	`group`
`/etc/group`	`group.byname`	`group`
`/etc/hosts`		`hosts`
`/etc/netgroup`	`netgroup`	`netgroup`

Local files	NIS maps	NIS+ tables
`/etc/netgroup`	`netgroup.byhost`	`netgroup`
`/etc/netgroup`	`netgroup.byuser`	`netgroup`
		`cred`
`/etc/networks`		`networks`
`/etc/passwd`	`passwd.byname`	`passwd`
`/etc/passwd`	`passwd.byuid`	`passwd`
`/etc/protocols`		`protocols`
`/etc/rpc`		`rpc`
`/etc/services`		`services`

4.22.2 NIS to NIS+ Command Compatibility

The following table lists the NIS and NIS+ commands supported on a UNICOS system.

Table 13. Comparing NIS and NIS+ commands on a UNICOS system

NIS server	NIS-compatible server	NIS+ server
`makeddm`	`---`	`nistbladm`, `nisaddent`
`ypbind`	`ypbind`	`---`
`ypcat`	`ypcat`	`niscat`
`ypwhich -m`	`ypwhich -m`	`niscat -o`
`ypinit -m`	`ypinit -c`	`---`
`ypinit -s`	`---`	`---`
`ypmake`	`---`	`nissetup`, `nisaddent`
`ypmatch`	`---`	`nismatch`, `nisgrep`
`yppasswd`	`---`	`nispasswd`
`yppush`	`---`	`nisping`
`yppoll`	`---`	`---`
`ypserv`	`nisd -Y`	`nisd`

NIS server	NIS-compatible server	NIS+ server
`ypset`	`---`	`---`
`ypxfr`	`---`	`---`

MIB Variables Supported by Cray [A]

This appendix lists the management information base (MIB) variables that Cray supports for the Simple Network Management Protocol (SNMP).

Note: SNMP is not part of the Cray ML-Safe configuration of the UNICOS system.

A.1 System Group

Cray supports the following system group (1.3.6.1.2.1.1) variables:

```
sysContact
sysDescr
sysLocation
sysName
sysObjectID
sysServices
sysUpTime
```

A.2 Interface Group

Cray supports the following interface group (1.3.6.1.2.1.2) variables:

```
ifAdminStatus
ifDescr
ifInDiscards
ifInErrors
ifInNUcastPkts
ifInOctets
ifInUcastPkts
ifInUnknownProtos
ifIndex
ifLastChange
ifMtu
ifNumber
ifOperStatus
ifOutDiscards
ifOutErrors
```

```
ifOutNUcastPkts
ifOutOctets
ifOutQLen
ifOutUcastPkts
ifPhysAddress
ifSpecific
ifSpeed
ifType
```

A.3 Address Translation Group

Cray supports the following address translation group (1.3.6.1.2.1.3) variables:

```
atIfIndex
atNetAddress
atPhysAddress
```

A.4 IP Group

Cray supports the following IP group (1.3.6.1.2.1.4) variables:

```
ipAdEntAddr
ipAdEntBcastAddr
ipAdEntIfIndex
ipAdEntNetMask
ipAdEntReasmMaxSize
ipDefaultTTL
ipForwDatagrams
ipForwarding
ipFragCreates
ipFragFails
ipFragOKs
ipInAddrErrors
ipInDelivers
ipInDiscards
ipInHdrErrors
ipInReceives
ipInUnknownProtos
ipNetToMediaIfIndex
ipNetToMediaNetAddress
ipNetToMediaPhysAddress
ipNetToMediaType
```

```
ipOutDiscards
ipOutNoRoutes
ipOutRequests
ipReasmFails
ipReasmOKs
ipReasmReqds
ipReasmTimeout
ipRouteAge
ipRouteDest
ipRouteIfIndex
ipRouteInfo
ipRouteMask
ipRouteMetric1
ipRouteMetric2
ipRouteMetric3
ipRouteMetric4
ipRouteMetric5
ipRouteNextHop
ipRouteProto
ipRouteType
ipRoutingDiscards
```

A.5 ICMP Group

Cray supports the following ICMP group (1.3.6.1.2.1.5) variables:

```
icmpInAddrMaskReps
icmpInAddrMasks
icmpInDestUnreachs
icmpInEchoReps
icmpInEchos
icmpInErrors
icmpInMsgs
icmpInParmProbs
icmpInRedirects
icmpInSrcQuenchs
icmpInTimeExcds
icmpInTimestampReps
icmpInTimestamps
icmpOutAddrMaskReps
icmpOutAddrMasks
icmpOutDestUnreachs
```

```
icmpOutEchoReps
icmpOutEchos
icmpOutErrors
icmpOutMsgs
icmpOutParmProbs
icmpOutRedirects
icmpOutSrcQuenchs
icmpOutTimeExcds
icmpOutTimestampReps
icmpOutTimestamps
```

A.6 TCP Group

Cray supports the following TCP group (1.3.6.1.2.1.6) variables:

```
tcpActiveOpens
tcpAttemptFails
tcpConnLocalAddress
tcpConnLocalPort
tcpConnRemAddress
tcpConnRemPort
tcpConnState
```

Note: Cray does not support the set operation on `tcpConnState`.

```
tcpCurrEstab
tcpEstabResets
tcpInErrs
tcpInSegs
tcpMaxConn
tcpOutRsts
tcpOutSegs
tcpPassiveOpens
tcpRetransSegs
tcpRtoAlgorithm
tcpRtoMax
tcpRtoMin
```

A.7 UDP Group

Cray supports the following UDP group (1.3.6.1.2.1.7) variables:

```
udpInDatagrams
udpInErrors
udpLocalAddress
udpLocalPort
udpNoPorts
udpOutDatagrams
```

A.8 SNMP Group

Cray supports the following SNMP group (1.3.6.1.2.1.8) variables:

```
snmpEnableAuthenTraps
snmpInASNParseErrs
snmpInBadCommunityNames
snmpInBadCommunityUses
snmpInBadValues
snmpInBadVersions
snmpInGenErrs
snmpInGetNexts
snmpInGetRequests
snmpInGetResponses
snmpInNoSuchNames
snmpInPkts
snmpInReadOnlys
snmpInSetRequests
snmpInTooBigs
snmpInTotalReqVars
snmpInTotalSetVars
snmpInTraps
snmpOutBadValues
snmpOutGenErrs
snmpOutGetNexts
snmpOutGetRequests
snmpOutGetResponses
snmpOutNoSuchNames
snmpOutPkts
snmpOutSetRequests
snmpOutTooBigs
snmpOutTraps
```

Index