Administering TCP/IP Networks, IPMP, and IP Tunnels in Oracle® Solaris 11.3

Part No: E54741
November 2016

Administering TCP/IP Networks, IPMP, and IP Tunnels in Oracle Solaris 11.3

Part No: E54741

Contents

Using This Documentation

- **Overview** – Describes tasks for administering TCP/IP networks, IPMP, and IP tunnels in the Oracle Solaris operating system (OS).
- **Audience** – System administrators.
- **Required knowledge** – Advanced experience with network administration, including administering TCP/IP networks and advanced networking features such as IPMP and IP tunnels.

Product Documentation Library

Documentation and resources for this product and related products are available at `http://www.oracle.com/pls/topic/lookup?ctx=E53394`.

Feedback

Provide feedback about this documentation at `http://www.oracle.com/goto/docfeedback`.

♦ ♦ ♦ CHAPTER 1

Administering TCP/IP Networks

This chapter describes how to administer the TCP/IP protocol on systems that have Oracle Solaris installed. The tasks that are in this chapter assume that you have an operational TCP/IP network at your site, either IPv4-only or a dual-stack IPv4/IPv6 network.

For information about planning your network deployment, see *Planning for Network Deployment in Oracle Solaris 11.3*.

For network configuration tasks, see *Configuring and Managing Network Components in Oracle Solaris 11.3*.

For information about some of the commands that are described in this chapter to observe network traffic usage at the different layers of the Oracle Solaris network protocol stack, see Chapter 2, "Using Observability Tools to Monitor Network Traffic Usage" in *Troubleshooting Network Administration Issues in Oracle Solaris 11.3*.

This chapter contains the following topics:

Customizing TCP/IP Properties

You use the `ipadm` command to configure the majority of TCP/IP properties, also known as *tunables*. The `ipadm` command replaces the `ndd` command as the primary tool for setting tunables. For more information about these changes, see "Comparing the ndd Command to the ipadm Command" in *Transitioning From Oracle Solaris 10 to Oracle Solaris 11.3*.

TCP/IP properties can be either interface-based or global and properties can be applied to a specific interface or globally to all of the interfaces within a zone. Global properties can also have different values within different non-global zones. For a complete list of the supported protocol properties, see the `ipadm`(1M) man page.

Typically, the default values of the TCP/IP protocol suffice for the network to function. However, if these values are insufficient for your particular network topology, you can customize the properties by using the following three `ipadm` subcommands:

- `ipadm show-prop -p` *property protocol* – Displays the properties of a protocol and its current values. If you do not use the `-p` *property* option, then all of the properties of the protocol are displayed. If you do not specify a protocol, then all of the properties of all of the protocols are displayed.
- `ipadm set-prop -p` *property*=*value protocol* – Assigns a value to the protocol's property.
  - To assign multiple values to a protocol's property, use the following syntax:

    `ipadm set-prop [-t] -p` *property*=*value*`[,...] protocol`
  - To remove one value from a set of values for a given property, use the `-=` qualifier:

    `ipadm set-prop -p` *property*-=*value2*
- Reset a specific protocol property to its default values as follows:

  `ipadm reset-prop -p` *property protocol*

For information about customizing IP interface properties, see "Customizing IP Interface Properties and Addresses" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.

For information about customizing IP address properties, see "Customizing IP Address Properties" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.

Enabling Packet Forwarding Globally

When you enable forwarding on an IP interface by using the `ipadm set-ifprop` command, it is only enabled for that interface and forwarding on all of the other interfaces remains unchanged. Setting packet forwarding on an individual IP interface property enables you to implement this feature selectively on specific interfaces on the system. For more information, see "Enabling Packet Forwarding" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.

To enable packet forwarding on an entire system, regardless of the number of IP interfaces, use the `protocol` property. The `forwarding` property is the global IP property that is used to manage forwarding, which is the same property name that is used to manage forwarding on individual IP interfaces. Because you can enable forwarding for either IPv4 or IPv6 protocols (or both), you must manage each protocol individually.

For example, you would enable packet forwarding for all IPv4 and IPv6 traffic on the system as follows:

```
# ipadm show-prop -p forwarding ip
PROTO   PROPERTY     PERM   CURRENT   PERSISTENT   DEFAULT   POSSIBLE
ipv4    forwarding   rw     off       --           off       on,off
ipv6    forwarding   rw     off       --           off       on,off

# ipadm set-prop -p forwarding=on ipv4
# ipadm set-prop -p forwarding=on ipv6

# ipadm show-prop -p forwarding ip
PROTO   PROPERTY     PERM   CURRENT   PERSISTENT   DEFAULT   POSSIBLE
ipv4    forwarding   rw     on        on           off       on,off
ipv6    forwarding   rw     on        on           off       on,off
```

Note - The `forwarding` property of either IP interfaces or protocols is not exclusive. You can set the property for the interface and the protocol at the same time. For example, you could enable packet forwarding globally on the protocol, and then customize packet forwarding for each IP interface on the system. Thus, although enabled globally, you can still manage packet forwarding selectively on a per-interface basis.

Administering Default Address Selection

Oracle Solaris enables a single interface to have multiple IP addresses. For example, technologies such as IPMP enable multiple network interface cards (NICs) to connect to the same IP link layer. That link can have one or more IP addresses. Additionally, interfaces on IPv6-enabled systems have a link-local IPv6 address, at least one IPv6 routing address, and an IPv4 address for at least one interface.

When the system initiates a transaction, an application makes a call to the `getaddrinfo` socket. The `getaddrinfo` socket discovers the possible address that is in use on the destination system. The kernel then prioritizes this list to find the best destination to use for the packet. This process is called *destination address ordering*. The Oracle Solaris kernel then selects the appropriate format for the source address, given the best destination address for the packet. This process is known as *address selection*. For more information on destination address ordering, see the `getaddrinfo`(3SOCKET) man page.

Both IPv4-only and dual-stack IPv4/IPv6 systems must perform default address selection. In most circumstances, you do not need to change the default address selection mechanisms. However, you might need to change the priority of address formats to support IPMP or to prefer 6to4 address formats, for example.

Description of the `/etc/inet/ipaddrsel.conf` Configuration File

The `/etc/inet/ipaddrsel.conf` file contains the IPv6 default address selection policy table. When you install Oracle Solaris with IPv6 enabled, this file contains the contents that are shown in Table 1, "IPv6 Address Selection Policy Table," on page 14.

You can edit the contents of `/etc/inet/ipaddrsel.conf`. However, in most cases, you should refrain from modifying this file. If modification is necessary, refer to the procedure "How to Administer the IPv6 Address Selection Policy Table" on page 15. For more information on `ippaddrsel.conf`, refer to "Reasons to Modify the IPv6 Address Selection Policy Table" on page 15 and the `ipaddrsel.conf`(4) man page.

Description of the `ipaddrsel` Command

The `ipaddrsel` command enables you to modify the IPv6 default address selection policy table.

The Oracle Solaris kernel uses the IPv6 default address selection policy table to perform destination address ordering and source address selection for an IPv6 packet header. The `/etc/inet/ipaddrsel.conf` file contains the policy table.

The following table lists the default address formats and their priorities for the policy table. You can find technical details for IPv6 address selection in the `inet6`(7P) man page.

TABLE 1 IPv6 Address Selection Policy Table

Prefix	Precedence	Definition
`::1/128`	50	Loopback
`::/0`	40	Default
`2002::/16`	30	6to4
`::/96`	20	IPv4 Compatible

Prefix	Precedence	Definition
`::ffff:0:0/96`	10	IPv4

In this table, IPv6 prefixes (`::1/128` and `::/0`) take precedence over 6to4 addresses (`2002::/16`) and IPv4 addresses (`::/96` and `::ffff:0:0/96`). Therefore, by default, the kernel selects the global IPv6 address of the interface for packets going to another IPv6 destination. The IPv4 address of the interface has a lower priority, particularly for packets going to an IPv6 destination. Given the selected IPv6 source address, the kernel also uses the IPv6 format for the destination address.

Reasons to Modify the IPv6 Address Selection Policy Table

Under most instances, you do not need to change the IPv6 default address selection policy table. If you do need to administer the policy table, use the `ipaddrsel` command.

You might want to modify the policy table for the following reasons:

- If the system has an interface that is used for a 6to4 tunnel, you can assign a higher priority to the 6to4 addresses.
- If you want a particular source address to be used only in communications with a particular destination address, you can add these addresses to the policy table. Then, you can use the `ipadm` command to flag these addresses, as preferred. For more information, see the `ipadm`(1M) man page.
- If you want IPv4 addresses to take precedence over IPv6 addresses, you can change the priority of `::ffff:0:0/96` to a higher number.
- If you need to assign a higher priority to deprecated addresses, you can add the deprecated address to the policy table. For example, site-local addresses are now deprecated in IPv6. These addresses have the prefix `fec0::/10`. You can change the policy table to assign a higher priority to site-local addresses.

For details about the `ipaddrsel` command, see the `ipaddrsel`(1M) man page.

▼ How to Administer the IPv6 Address Selection Policy Table

The following procedure describes how to modify the address selection policy table. For conceptual information about IPv6 default address selection, see “Description of the ipaddrsel Command” in *Administering TCP/IP Networks, IPMP, and IP Tunnels in Oracle Solaris 11.3*.

Caution - Do not change the IPv6 address selection policy table except for the reasons that are provided in the following procedure. Doing so, can cause problems on the network due to a badly constructed policy table. Also, be sure to save a backup copy of the policy table, as shown in this procedure.

1. **Become an administrator.**

2. **Review the current IPv6 address selection policy table.**

```
# ipaddrsel
# Prefix                    Precedence Label
::1/128                             50 Loopback
::/0                                40 Default
2002::/16                           30 6to4
::/96                               20 IPv4_Compatible
::ffff:0.0.0.0/96                   10 IPv4
```

3. **Make a backup copy of the default address policy table.**

```
# cp /etc/inet/ipaddrsel.conf /etc/inet/ipaddrsel.conf.orig
```

4. **Add any customizations to the `/etc/inet/ipaddrsel.conf` file.**

```
# pfedit /etc/inet/ipaddrsel.conf
```

   Use the following syntax for entries in `/etc/inet/ipaddrsel`:

   prefix/prefix-length precedence label [# comment]

   See Example 1, "Modifying the Default Pv6 Address Selection Policy Table," on page 16 for examples of some common modifications that you might make.

5. **Load the modified policy table into the kernel.**

```
# ipaddrsel -f /etc/inet/ipaddrsel.conf
```

6. **If the modified policy table has problems, restore the default IPv6 address selection policy table.**

```
# ipaddrsel -d
```

Example 1 Modifying the Default Pv6 Address Selection Policy Table

The following are some common modifications that you might want to make to your policy table:

- Assign the highest priority to 6to4 addresses.

```
2002::/16                          50 6to4
```

```
::1/128                           45 Loopback
```

The 6to4 address format now has the highest priority, 50. Loopback, which previously had a 50 precedence, now has a 45 precedence. The other addressing formats remain the same.

- Designate a specific source address to be used in communications with a specific destination address.

```
::1/128                           50 Loopback
2001:1111:1111::1/128             40 ClientNet
2001:2222:2222::/48               40 ClientNet
::/0                              40 Default
```

This particular entry is useful for hosts with only one physical interface. Here, `2001:1111:1111::1/128` is preferred as the source address on all packets that are bound for destinations within network `2001:2222:2222::/48`. The 40 priority gives higher precedence to the source address `2001:1111:1111::1/128` than to other address formats configured for the interface.

- Favor IPv4 addresses over IPv6 addresses.

```
::ffff:0.0.0.0/96                 60 IPv4
::1/128                           50 Loopback
.
.
```

The IPv4 format `::ffff:0.0.0.0/96` has its precedence changed from the default 10 to 60, the highest priority in the table.

▼ How to Modify the IPv6 Address Selection Table for the Current Session Only

When you edit the `/etc/inet/ipaddrsel.conf` file, any modifications you make persist across system reboots. If you want the modified policy table to exist only in the current session, use the following procedure.

1. **Become an administrator.**

2. **Copy the contents of the `/etc/inet/ipaddrsel` file to *filename*, where *filename* represents any name that you choose.**

```
# cp /etc/inet/ipaddrsel filename
```

3. **Use the `pfedit` command to edit the policy table in *filename* to your specifications.**

4. **Load the modified policy table into the kernel.**

```
# ipaddrsel -f filename
```

The kernel uses the new policy table until you reboot the system.

Administering Transport Layer Services

This section contains the following topics:

- "Recommendations for Configuring Systems That Run `inetd` Based Services" on page 18
- "Adding `inetd` Based Services That Run Over Transport Layer Protocols" on page 20
- "Setting Up a Privileged Port" on page 23
- "Implementing Traffic Congestion Control" on page 24
- "Logging IP Addresses of All Incoming TCP Connections" on page 26
- "Configuring TCP Wrappers" on page 26
- "Changing the TCP Receive Buffer Size" on page 27

The following transport layer protocols are a standard part of Oracle Solaris: Transmission Control Protocol (TCP), Stream Control Transmission Protocol (SCTP), and User Datagram Protocol (UDP). These protocols typically need no intervention to run properly. However, circumstances at your site might require you to log or modify services that run over these transport layer protocols. In this case, you must modify the profiles for these services or add a new service instance by using Service Management Facility (SMF) commands. For more information, see "Modifying Services that are Controlled by inetd" in *Managing System Services in Oracle Solaris 11.3* and "Adding `inetd` Based Services That Run Over Transport Layer Protocols" on page 20.

The `inetd` daemon is responsible for starting standard Internet services when a system boots. These services include applications that use TCP, SCTP, or UDP as the transport layer protocol. You can modify the existing Internet services or add new services by using SMF commands. For more information, see the `inetd`(1M) man page. For recommendations on configuring systems that use `inetd` processes, see "Recommendations for Configuring Systems That Run `inetd` Based Services" on page 18.

Recommendations for Configuring Systems That Run `inetd` Based Services

To safeguard against potential security vulnerabilities, you should configure systems that are running necessary `inetd` based services so that the number of concurrent processes are limited. In addition, if a service that is controlled by the `inetd` daemon is not required disable that service. See "Stopping a Service" in *Managing System Services in Oracle Solaris 11.3* for instructions.

The `inetd` daemon is the delegated restarter for Internet services for the Service Management Facility (SMF). Use the `inetadm` command to display a list of the services that are controlled by `inetd`:

```
$ inetadm
ENABLED   STATE         FMRI
disabled  disabled      svc:/application/cups/in-lpd:default
enabled   online        svc:/network/finger:default
disabled  disabled      svc:/application/x11/xvnc-inetd:default
```

Then, decide on the desired maximum number of concurrent processes for a given service and set allowable limits for that service by using the `inetadm` command to set the `max_copies` property for the specified service.

For example, you would set a limit of concurrent instances for the `finger` service to `3` as follows:

```
# inetadm -m svc:/network/finger:default max_copies=3
```

Then, verify the change as follows:

```
$ inetadm -l finger | grep copies
max_copies=3
```

The `-l` option displays the current values for all of the properties of a specified service.

Use the `-p` option to list the properties that are common to all services that are managed by the `inetd` daemon and their default values.

```
$ inetadm -p
NAME=VALUE
bind_addr=""
bind_fail_max=-1
bind_fail_interval=-1
max_con_rate=-1
max_copies=-1
con_rate_offline=-1
failrate_cnt=40
failrate_interval=60
inherit_env=TRUE
tcp_trace=FALSE
tcp_wrappers=FALSE
connection_backlog=10
tcp_keepalive=FALSE
```

The `-1` value for the `max_copies` property means that by default the number of processes that can run concurrently is not limited.

Note - Because it is difficult to determine a limit that is appropriate for all customers and all environments, Oracle does not provide a generic default limit recommendation for this value.

Use the `-m` option to modify the value of a property for a specified service. For example, you would limit the number of `finger` processes that can run concurrently to `5` as follows:

```
$ inetadm -m svc:/network/finger:default max_copies=5
$ inetadm -l finger | grep copies
max_copies=5
```

For more information, see the `inetadm`(1M) man page.

Adding `inetd` Based Services That Run Over Transport Layer Protocols

In Oracle Solaris, several transport layer protocols, for example, SCTP, TCP, and UDP, provide services to application layer protocols. By default, these protocols typically do not require any additional configuration. However, you might need to explicitly configure certain application layer services to use certain network protocols. Some examples include the `echo` and `discard` applications.

▼ How to Add an `inetd` Based Service That Runs Over a Transport Layer Protocol

The following procedure describes the basic steps for adding `inet` services (those controlled by the `inetd` daemon) for a transport layer protocol to the SMF repository. For more information about modifying services that are controlled by `inetd`, see "Modifying Services that are Controlled by inetd" in *Managing System Services in Oracle Solaris 11.3*.

For example purposes only, the procedure describes how to add a new SCTP `echo` service by using the manifest `echo.xml` (`/lib/svc/manifest/network/echo.xml`). However, you can use the same basic procedure to add different services that run over other transport layer protocols, for example, TCP and UDP.

Before You Begin Prior to performing this procedure, confirm that the service you want to add is controlled by the `inetd` daemon. For example, the output of the following command shows that the `echo` service, which is used in this procedure's examples, is controlled by `inetd`:

```
$ inetadm
.
.
.
```

```
disabled disabled       svc:/network/echo:dgram
disabled disabled       svc:/network/echo:stream
.
.
.
```

1. **Log in to the local system with a user account that has write privileges for system files.**

2. **Add a definition for the new service instance to the `/etc/services` file by using the `pfedit` command.**

   See the `pfedit`(1M) man page.

   Use the following syntax for the service definition:

   service-name port/protocol aliases

3. **Determine the location of the manifest for the specified service.**

   ```
   $ svcs -l service-name
   ```

   For example, you would locate the manifest for the `echo` service as follows:

   ```
   $ svcs -l echo
   fmri         svc:/network/echo:stream
   name         echo
   enabled      false
   state        disabled
   next_state   none
   state_time   Thu Dec  3 08:11:11 2015
   restarter    svc:/network/inetd:default
   manifest     /etc/svc/profile/generic.xml
   manifest     /lib/svc/manifest/network/echo.xml

   fmri         svc:/network/echo:dgram
   name         echo
   enabled      false
   state        disabled
   next_state   none
   state_time   Thu Dec  3 08:11:10 2015
   restarter    svc:/network/inetd:default
   manifest     /etc/svc/profile/generic.xml
   manifest     /lib/svc/manifest/network/echo.xml
   ```

4. **Using a text editor, modify the existing service manifest to add a new instance with the appropriate property values.**

   For example, `/lib/svc/manifest/network/echo.xml` is where the `echo` service manifest is located.

   a. **Change to the directory where the service manifest is stored.**

```
# cd /lib/svc/manifest/network
```

b. **Copy and paste an existing service instance element.**

For this particular example, you could use either the `dgram` or `stream` element.

c. **Provide a different name for the service instance element (`sctp_stream` is used in this example), then change the property value as appropriate.**

In this example, the value of the `proto` property is changed to `sctp`.

5. **Edit the `/etc/svc/profile/generic.xml` profile file to add the new service instance.**

For example, you would add a new `sctp_stream` instance as follows:

```
<service name='network/echo' version='1' type='service'>
<instance name='stream' enabled='false'/>
<instance name='dgram' enabled='false'/>
<instance name='sctp_stream' enabled='false'/>
</service>
```

6. **Restart the `manifest-import` service.**

```
$ svcadm restart manifest-import
```

7. **Verify that the new service instance has been added.**

```
$ svcs FMRI
```

For the *FMRI* argument, use the Fault Managed Resource Identifier (FMRI) of the service manifest for example, the `echo` service's FMRI:

```
$ svcs echo
```

Or, you can use the `inetadm` command the display similar information:

```
$ inetadm -l echo
```

8. **Verify that the property values for the new instance (for example, `echo:sctp_stream`) are correct.**

```
$ inetadm -l echo:sctp_stream
SCOPE    NAME=VALUE
         name="echo"
         endpoint_type="stream"
         proto="sctp"
         isrpc=FALSE
         wait=FALSE
         exec="/usr/lib/inet/in.echod -s"
         user="root"
```

```
default  bind_addr=""
default  bind_fail_max=-1
default  bind_fail_interval=-1
default  max_con_rate=-1
default  max_copies=-1
default  con_rate_offline=-1
default  failrate_cnt=40
default  failrate_interval=60
default  inherit_env=TRUE
default  tcp_trace=FALSE
default  tcp_wrappers=FALSE
default  connection_backlog=10
default  tcp_keepalive=FALSE
```

9. **(Optional) Enable the new service instance.**

```
# inetadm -e FMRI
```

For example, you would enable the `echo` service as follows:

```
$ inetadm -e svc:/network/echo:sctp_stream
$ svcs echo
```

Or, use the `inetadm` command as follows:

```
$ inetadm | grep echo
```

Setting Up a Privileged Port

On transport protocols such as TCP, UDP, and SCTP, ports `1-1023` are by default privileged ports. To bind to a privileged port, a process must be running with `root` permissions. Ports that are greater than `1023` are by default non-privileged. You can use the `ipadm` command to extend the range of privileged ports, or you can mark specific ports in the non-privileged range as privileged ports.

To manage the range of privileged ports, you can customize the following transport protocol properties:

`smallest_nonpriv_port`	Specifies a value that indicates the beginning of the range of non-privileged port numbers, which are the ports to which regular users can bind. You can set individual ports within the non-privileged range as privileged ports. Use the `ipadm show-prop` command to display the property's values.
`extra_priv_ports`	Specifies which ports outside of the privileged range are also privileged. Use the `ipadm set-prop` command to specify ports that you want to restrict. You can assign multiple values to this property.

As an example, suppose you want to set TCP ports 3001 and 3050 as privileged ports, with access restricted to just the root role. The smallest_nonpriv_port property indicates that 1024 is the lowest port number for a non-privileged port. Therefore, you can change the designated ports 3001 and 3050 to privileged ports as follows:

```
# ipadm show-prop -p smallest_nonpriv_port tcp
PROTO PROPERTY                PERM   CURRENT   PERSISTENT   DEFAULT   POSSIBLE
tcp   smallest_nonpriv_port   rw     1024      --           1024      1024-32768

# ipadm show-prop -p extra_priv_ports tcp
PROTO   PROPERTY           PERM   CURRENT     PERSISTENT   DEFAULT     POSSIBLE
tcp     extra_priv_ports   rw     2049,4045   --           2049,4045   1-65535

# ipadm set-prop -p extra_priv_ports+=3001 tcp
# ipadm set-prop -p extra_priv_ports+=3050 tcp
# ipadm show-prop -p extra_priv_ports tcp
PROTO   PROPERTY           PERM   CURRENT     PERSISTENT   DEFAULT     POSSIBLE
tcp     extra_priv_ports   rw     2049,4045   3001,3050    2049,4045   1-65535
                                  3001,3050
```

You would remove a privileged port, for example 4045, as follows:

```
# ipadm set-prop -p extra_priv_ports-=4045 tcp
# ipadm show-prop -p extra_priv_ports tcp
PROTO   PROPERTY           PERM   CURRENT     PERSISTENT   DEFAULT     POSSIBLE
tcp     extra_priv_ports   rw     2049,3001   3001,3050    2049,4045   1-65535
                                  3050
```

Implementing Traffic Congestion Control

Network congestion typically occurs in the form of router buffer overflows, when nodes send more packets than the network can accommodate. Various algorithms prevent traffic congestion through establishing controls on the sending systems. These algorithms are supported in Oracle Solaris and can be easily added or directly plugged into the OS, as shown in the following table that describes the supported built-in algorithms.

Algorithm	Oracle Solaris Name	Description
NewReno	newreno	Default algorithm in Oracle Solaris. This control mechanism includes a sender's congestion window, slow start, and congestion avoidance.
HighSpeed	highspeed	One of the best known and simplest modifications of NewReno for high-speed networks.
CUBIC	cubic	Currently the default algorithm in Linux 2.6. Changes the congestion avoidance phase from linear window increase to a cubic function.

Algorithm	Oracle Solaris Name	Description
Vegas	`vegas`	A classic delay-based algorithm that attempts to predict congestion without triggering actual packet loss.

Congestion control is enabled by setting the following control-related TCP properties. Although these properties are listed for TCP, the control mechanism that is enabled by these properties also applies to SCTP traffic.

`cong_enabled`	Contains a list of algorithms, separated by commas, that are currently operational on the system. You can add or remove algorithms to enable only those algorithms that you want to use. This property can have multiple values. Therefore you must use either the += qualifier or the `-=` qualifier, depending on the change that you want to make.
`cong_default`	Used by default when applications do not specify the algorithms explicitly in socket options. The value of the `cong_default` property applies to both global and non-global zones.

The following example shows how you would add an algorithm for congestion control to the protocol:

```
# ipadm set-prop -p cong_enabled+=algorithm tcp
```

Remove an algorithm as follows:

```
# ipadm set-prop -p cong_enabled-=algorithm tcp
```

Replace the default algorithm as follows:

```
# ipadm set-prop -p cong_default=algorithm tcp
```

Note - No sequence rules are followed when you add or remove algorithms. You can remove an algorithm before adding other algorithms to a property. However, the `cong_default` property must always have a defined algorithm.

The following example shows how you might implement congestion control. In this example, the default algorithm for the TCP protocol is changed from `newreno` to `cubic` Then, the `vegas` algorithm is removed from the list of enabled algorithms.

```
# ipadm show-prop -p extra_priv_ports tcp
PROTO PROPERTY              PERM CURRENT      PERSISTENT   DEFAULT      POSSIBLE
tcp   extra_priv_ports      rw   2049,4045    --           2049,4045    1-65535

# ipadm show-prop -p cong_default,cong_enabled tcp
PROTO PROPERTY              PERM CURRENT      PERSISTENT   DEFAULT      POSSIBLE
tcp   cong_default          rw   newreno      --           newreno      newreno,cubic,
                                                                        highspeed,vegas
```

```
tcp   cong_enabled          rw   newreno,cubic, newreno,cubic, newreno  newreno,cubic,
                                 highspeed,    highspeed,               highspeed,vegas
                                 vegas         vegas

# ipadm set-prop -p cong_enabled-=vegas tcp
# ipadm set-prop -p cong_default=cubic tcp

# ipadm show-prop -p cong_default,cong_enabled tcp
PROTO PROPERTY              PERM CURRENT      PERSISTENT   DEFAULT      POSSIBLE
tcp   cong_default          rw   cubic        cubic        newreno      newreno,cubic,
                                                                        highspeed
tcp   cong_enabled          rw   newreno,cubic, newreno,cubic, newreno  newreno,cubic,
                                 highspeed    highspeed                 highspeed,vegas
```

Logging IP Addresses of All Incoming TCP Connections

IP addresses for all incoming TCP connections are logged by the `syslog` service, at the `daemon.notice` facility and level. The information is located in `/var/adm/messages`, unless you changed the local `syslog.conf` file setup. Note that changing the `syslog.conf` file setup can affect where this information is stored. For more details, see the `syslog.conf`(4) man page.

Set TCP tracing to `TRUE` (enabled) for all services that are managed by the `inetd` daemon as follows:

```
# inetadm -M tcp_trace=TRUE
```

Configuring TCP Wrappers

TCP wrappers add a measure of security for service daemons by standing between the daemon and incoming service requests. TCP wrappers log successful and unsuccessful connection attempts. Additionally, TCP wrappers can provide access control, allowing or denying the connection, depending on where the request originates. You can use TCP wrappers to protect daemons such as Secure Shell (SSH), Telnet, and the File Transfer Protocol (FTP). For information about TCP wrappers and `sendmail`, see the `sendmail`(1M) man page.

▼ How to Use TCP Wrappers to Control Access to TCP Services

The `tcpd` program implements *TCP wrappers*.

1. **Become the `root` role.**

2. **Set TCP wrappers to enabled.**

```
# inetadm -M tcp_wrappers=TRUE
```

3. **Configure the TCP wrappers access control policy.**

   For instructions, refer to the `hosts_access`(3) man page, which can be found in the `/usr/sfw/man` directory.

Changing the TCP Receive Buffer Size

The size of the TCP receive buffer is set by using the `recv_buf` TCP property, which is 128 KB by default. However, applications do not use available bandwidth uniformly. Thus, connection latency might require you to change the default size. For example, using the Secure Shell feature of Oracle Solaris causes overhead on bandwidth use because of the additional checksum and encryption processes that are performed on the data stream. Thus, the buffer size might need to be increased. Likewise, to enable applications that perform bulk transfers to use bandwidth efficiently, the same buffer size adjustment is also required.

You can calculate the correct receive buffer size to use by estimating the bandwidth delay product (BDP). To calculate BDP, multiply the available bandwidth by the value of the connection latency.

Use the `ping -s` *host* command to obtain the value of the connection latency.

The appropriate receive buffer size approximates the value of the BDP. However, the use of bandwidth also depends on a variety of conditions. A shared infrastructure or the number of applications and users that compete for the use of bandwidth can change that estimate.

Change the value of the buffer size as follows:

```
# ipadm set-prop -p recv_buf=value tcp
```

The following example shows how to increase the buffer size to 164 KB:

```
# ipadm show-prop -p recv_buf tcp
PROTO PROPERTY   PERM CURRENT   PERSISTENT   DEFAULT  POSSIBLE
tcp   recv_buf   rw   128000       --        128000   2048-1048576

# ipadm set-prop -p recv_buf=164000 tcp

# ipadm show-prop -p recv_buf tcp
PROTO PROPERTY   PERM CURRENT   PERSISTENT   DEFAULT  POSSIBLE
tcp   recv_buf   rw   164000    164000       128000   2048-1048576
```

No set value for the buffer size is preferred because the preferred size varies depending on the circumstance. Consider the following examples where different values are set for the BDP in each network with specific conditions:

Typical 1 Gbps local area network (LAN) where 128 KB is the default value of the buffer size:	`BDP = 128 MBps * 0.001 s = 128 kB`
Theoretical 1Gbps wide area network (WAN) with 100 ms latency:	`BDP = 128 MBps * 0.1 s = 12.8 MB`
Europe-to-U.S. link (bandwidth measured by `uperf`)	`BDP = 2.6 MBps * 0.175 = 470 kB`

If you cannot compute the BDP, use the following guidelines:

- For bulk transfers over a LAN, the default value of the buffer size (128 KB) is sufficient.
- For most WAN deployments, the receive buffer size should be in the 2 MB range.

Caution - Increasing the TCP receive buffer size increases the memory footprint of many network applications.

Monitoring Network Status With the `netstat` Command

You use the `netstat` command to generate displays that show network status and protocol statistics. You can display the status of TCP, SCTP, and UDP endpoints in table format, as well as display routing table and interface information.

The `netstat` command displays various types of network data, depending on which command-line option you specify. The information that is displayed is particularly useful for system administration. The options that are used for some of the more commonly performed tasks are described in this chapter. For complete details, see the `netstat`(1M) man page.

This section contains the following topics:

- "Filtering `netstat` Output by Address Type" on page 29
- "Displaying the Status of Sockets" on page 29
- "Displaying Statistics by Protocol" on page 29
- "Displaying Network Interface Status" on page 32
- "Displaying User and Process Information" on page 33
- "Displaying the Status of Known Routes" on page 33
- "Displaying Additional Network Status With the `netstat` Command" on page 34

Filtering `netstat` Output by Address Type

You can use the `netstat` command to display both IPv4 and IPv6 network status. You can choose which protocol information to display by setting the `DEFAULT_IP` value in the `/etc/default/inet_type` file or by using the `-f` option. By permanently setting the `DEFAULT_IP` value, you can ensure that the `netstat` command displays only IPv4 information. To override this setting, use the `-f` option from the command line. For more information about the `inet_type` file, see the `inet_type`(4) man page.

Use can also use the `-f` option to specify the `inet`, `inet6`, `unix` (for Unix domain sockets that used for internal communication) or `sdp` (Socket Description Protocol) arguments.

Displaying the Status of Sockets

Use the `netstat` command to view the status of the sockets on a local host. You can specify several `netstat` command options as a regular user.

Display the status of connect sockets as follows:

```
% netstat
```

Display the status of all sockets, including unconnected listener sockets, as follows:

```
% netstat -a
```

EXAMPLE 2 Displaying Connected Sockets

The output of the `netstat` command shows extensive statistics. The following example shows how you would limit the output to IPv4 sockets only.

```
% netstat -f inet

TCP: IPv4
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q    State
-------------------- -------------------- ----- ------ ----- ------ -----------
system-1.ssh         remote.38474     128872      0 128872      0 ESTABLISHED
system2.40721        remote.ldap      49232       0 128872      0 ESTABLISHED
```

Displaying Statistics by Protocol

The `netstat -s` option displays protocol statistics for the UDP, TCP, SCTP, Internet Control Message Protocol (ICMP), and IP protocols.

Display protocol status as follows:

```
% netstat -s
```

Use the `-P` option to filter the output of the `netstat` command by protocol. This option is not limited to transport protocols.

You can specify the following values with this option:

- `icmp`
- `icmpv6`
- `igmp`
- `ipv6tcp`
- `rawip`
- `sctp`
- `tcp`
- `udp`

For example, you would filter the `netstat` output by the UDP protocol as follows:

```
# netstat -aP udp
UDP: IPv4

   Local Address   Remote Address   State  Send Buf  TxOverflows  Recv Buf   RxOverflows

---------------- --------------- ------- --------- ------------ --------     --------

      *.*                        Unbound    57344            0   57344              0

      *.*                        Unbound    57344            0   57344              0

      *.*                        Unbound    57344            0   57344              0

      *.*                        Unbound    57344            0   57344              0

...

      *.bootpc                      Idle    57344            0   57344              0

      *.dhcpv6-client               Idle    57344            0   57344              0

ip-10-134-63-206.bootpc             Idle    57344            0   57344              0

      *.sunrpc                      Idle    57344            0   57344              0

      *.*                        Unbound    57344            0   57344              0

      *.59730                       Idle    57344            0   57344              0

      *.sunrpc                      Idle    57344            0   57344              0
      *.*                           Unbound    57344           0   57344              0

      *.47158                          Idle    57344           0   57344              0

      *.*                           Unbound    57344           0   57344              0

      *.631                            Idle    57344           0   57344              0

      *.ntp                            Idle    57344           0   57344              0

      *.ntp                            Idle    57344           0   57344              0

localhost.ntp                          Idle    57344           0   57344              0

ip-10-134-63-206.ntp                   Idle    57344           0   57344              0

UDP: IPv6

   Local Address   Remote Address  State    If    Send Buf  TxOverflows  Recv Buf
 RxOverflows

-----------------   -------------  ----------   ----- -------- ----------- --------
 -------

      *.*                           Unbound            57344        0       57344
      0

      *.*                           Unbound            57344        0       57344
      0

      *.*                           Unbound            57344        0       57344
      0

      *.*                           Unbound            57344        0       57344
      0

      *.dhcpv6-client                  Idle            57344        0       57344
      0

...

localhost.ntp                          Idle            57344        0       57344
      0
```

As shown in the previous output, the `netstat -P udp` command displays the following four additional statistics in Oracle Solaris 12:

`Send buf`	Displays socket send buffer statistics.
`Recv buf`	Displays socket receive buffer statistics.

`TxOverflows`	Displays the number of times overflow for transmitting packets occurred. The counter is increased whenever IP cannot send the outgoing packet to the MAC layer due to unavailable space.
`RxOverflows`	Displays the number of times that an overflow for receiving packets occurred. The counter is increased whenever IP cannot send the incoming packet to the socket due to unavailable space The incoming packet is dropped in the case of an Rx overflow.

EXAMPLE 3 Displaying the Status of the TCP Protocol

The following example shows how you would display the results for the TCP protocol by specifying the `-P` option.

```
% netstat -P tcp

TCP: IPv4
Local Address     Remote Address               Swind Send-Q  Rwind  Recv-Q      State
----------------- --------------------         ----- ------  -----  ------     -------
lhost-1.login     abc.local.example.COM.980   49640      0    49640    0    ESTABLISHED
lhost.login       ghi.local.example.COM.1020  49640      1    49640    0    ESTABLISHED
remhost.1014      mno.remote.example.COM.nfsd 49640      0    49640    0      TIME_WAIT

TCP: IPv6
Local Address    Remote Address        Swind Send-Q Rwind Recv-Q   State If
--------------- --------------------- ------ ----- ------ ----------- -----
localhost.38983   localhost.32777       49152      0 49152      0 ESTABLISHED
localhost.32777   localhost.38983       49152      0 49152      0 ESTABLISHED
localhost.38986   localhost.38980       49152      0 49152      0 ESTABLISHED
```

Displaying Network Interface Status

Use the `i` option of the `netstat` command to display the state of the network interfaces that are configured on a local system. With this option, you can determine the number of packets a system transmits and receives on each network.

Display the status of the interfaces that are on a network as follows:

```
% netstat -i
```

The following example shows how you would use the `netstat` command with a filtering option to limit the output of a specific interface:

```
% netstat -i -I net0 -f inet
  Name  Mtu  Net/Dest        Address         Ipkts  Ierrs Opkts  Oerrs Collis Queue
  net0  1500 abc.example.com abc.example.com 231001 0     55856  0     0      0
```

EXAMPLE 4 Displaying Network Interface Status

The following example shows the status of IPv4 and IPv6 packet flow through the tems's interfaces. For example, the input packet count (`Ipkts`) that is displayed for a server can increase each time a client tries to boot, while the output packet count (`Opkts`) remains steady. This outcome suggests that the server is recognizing the boot request packets from the client. However, the server does not know to respond to them. This confusion might be caused by an incorrect address in the `hosts`, or `ethers` database.

If the input packet count is steady over time, then the tem does not see the packets at all. This outcome suggests a different type of failure, possibly a hardware problem.

```
Name  Mtu  Net/Dest      Address        Ipkts  Ierrs Opkts  Oerrs Collis Queue
lo0   8232 loopback      localhost      142    0     142    0     0      0
net0  1500 host58        host58        1106302 0     52419  0     0      0

Name  Mtu  Net/Dest      Address                    Ipkts  Ierrs Opkts  Oerrs Collis
lo0   8252 localhost     localhost                   142    0     142    0     0
net0  1500 fe80::a00:20ff:feb9:4c54/10 fe80::a00:20ff:feb9:4c54 1106305 0 52422 0  0
```

Displaying User and Process Information

The `netstat` command includes a `-u` option that you can use to obtain information about users and processes on the system that are using specific ports. Use the `netstat -u` command to display the user, process ID, and the program that created the network endpoint or the program that currently controls the network endpoint.

To produce better alignment of the `netstat` command output, you can specify the `-n` option with the `-u` option. For more information and detailed examples, see the `netstat`(1M) man page.

Displaying the Status of Known Routes

Use the `-r` option with the `netstat` command to display the routing table for a local tem. This table displays the status of all routes that are known to a stem.

```
% netstat -r
```

EXAMPLE 5 Displaying Routing Table Output With the `netstat` Command

The following example shows the output of the `netstat -r` command.

```
Routing Table: IPv4
Destination           Gateway           Flags  Ref   Use   Interface
```

```
-------------------- -------------------- ----- ----- ------ ---------
host15               myhost               U         1  31059  net0
10.0.0.14            myhost               U         1      0  net0
default              distantrouter        UG        1      2  net0
localhost            localhost            UH        42019361  lo0

Routing Table: IPv6
Destination/Mask      Gateway                           Flags Ref  Use   If
--------------------- ---------------------------          ----- --- ------ -----
2002:0a00:3010:2::/64 2002:0a00:3010:2:1b2b:3c4c:5e6e:abcd U     1   0      net0:1
fe80::/10             fe80::1a2b:3c4d:5e6f:12a2            U     1   23     net0
ff00::/8              fe80::1a2b:3c4d:5e6f:12a2            U     1   0      net0
default               fe80::1a2b:3c4d:5e6f:12a2            UG    1   0      net0
localhost             localhost                            UH    9   21832  lo0
```

The following table describes the information that is displayed in the output of the `netstat -r` command.

Parameter	Description
`Destination` `Destination/Mask`	Specifies the host that is the destination endpoint of the route. Note that the IPv6 routing table shows the prefix for a 6to4 tunnel endpoint (`2002:0a00:3010:2::/64`) as the route destination endpoint.
`Gateway`	Specifies the gateway to use for forwarding packets.
`Flags`	Indicates the current status of the route. The `U` flag indicates that the route is up. The `G` flag indicates that the route is to a gateway.
`Use`	Shows the number of packets sent.
`Interface`	Indicates the particular interface on the local host that is the source endpoint of the transmission.

Displaying Additional Network Status With the `netstat` Command

You can use the `netstat` command with various other command-line options to display additional network status, beyond what is documented in this chapter. There are 10 forms of the command and each form produces a different table of statistics about the various parts of the networking subsystem.

Some of the additional statistics that you can obtain include the following:

`netstat -g`	Displays multicast group memberships
`netstat -P`	Displays net-to-media tables: ARP and NDP mappings
`netstat -m`	Displays STREAMS memory statistics

`netstat -M`	Displays the multicast routing table
`netstat -D`	Displays the status of DHCP leases
`netstat -d`	Displays the destination cache table

For complete details, see the `netstat`(1M) man page.

Performing TCP and UDP Administration With the `netcat` Utility

Use the `netcat` (`nc`) utility to perform a variety of tasks that are associated with TCP or UDP administration. You can use the command for both IPv4 and IPv6 networks.

You can perform the following tasks by using the `netcat` utility:

- Open TCP connections
- Send UDP packets
- Listen on arbitrary TCP and UDP ports
- Perform port scanning

Unlike the `telnet` command, where each error message is emitted separately to standard output, error messages that are generated by `nc` scripts are combined into one standard error, which is a more efficient method.

The `netcat` utility supports a new `-M` option, which enables you to specify per socket Service Level Agreement (SLA) properties. When you specify the appropriate properties along with the `-M` option, a MAC flow for the socket is created. For example, you might use the `-M` option as follows:

```
% nc -M maxbw=50M host.example.com 7777
% nc -l -M priority=high,inherit=on 2222
```

As shown in the previous example, the `-M` option can be used to specify a comma-separated list of *name*=*value* pairs of SLA properties.

Some installation methods do not install the `netcat` software package by default. Check whether the package is installed on your system as follows:

```
% pkg list network/netcat
```

If the package is not installed, install it as follows:

```
% pfexec pkg install pkg:/network/netcat
```

For more details, see the `netcat`(1) man page.

Administering and Logging Network Status Displays

The following tasks describe how to check the status of the network by using well-known networking commands:

- "How to Control the Display Output of IP-Related Commands" on page 36
- "Logging Actions of the IPv4 Routing Daemon" on page 37
- "How to Trace the Activities of the IPv6 Neighbor Discovery Daemon" on page 38

▼ How to Control the Display Output of IP-Related Commands

You can control the output of the `netstat` command to display IPv4 information only or both IPv4 and IPv6 information.

1. **Create the `/etc/default/inet_type` file.**
2. **Add one of the following entries to the `/etc/default/inet_type` file, as required for your network:**

   - **To display IPv4 information only, set the following variable:**

     ```
     DEFAULT_IP=IP_VERSION4
     ```

   - **To display both IPv4 and IPv6 information, set one of the following variables:**

     ```
     DEFAULT_IP=BOTH
     ```

     ```
     DEFAULT_IP=IP_VERSION6
     ```

   For more information, see the `inet_type(4)` man page.

   Note - The `-f` option of the `netstat` command overrides the values that are set in the `inet_type` file.

Example 6 Controlling Output to Display IPv4 and IPv6 Information

The following example shows the output when you specify the `DEFAULT_IP=BOTH` or `DEFAULT_IP=IP_VERSION6` variable in the `inet_type` file:

```
# ifconfig -a
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
```

```
net0: flags=100001004843<UP,BROADCAST,RUNNING,MULTICAST,DHCP,IPv4,PHYSRUNNING> mtu 1500
 index 603
        inet 10.46.86.54 netmask ffffff00 broadcast 10.46.8.255
        ether 0:1e:68:49:98:80
lo0: flags=2002000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6,VIRTUAL> mtu 8252 index 1
        inet6 ::1/128
net0: flags=120002000841<UP,RUNNING,MULTICAST,IPv6,PHYSRUNNING> mtu 1500 index 603
        inet6 fe80::21e:68ff:fe49:9880/10
        ether 0:1e:68:49:98:80
net0:1: flags=120002080841<UP,RUNNING,MULTICAST,ADDRCONF,IPv6,PHYSRUNNING> mtu 1500
 index 603
        inet6 2001:b400:414:6059:21e:68ff:fe49:9880/64
```

When you specify the `DEFAULT_IP=IP_VERSION4` variable in the `inet_type` file, you should see the following output:

```
# ifconfig -a
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
net0: flags=100001004843<UP,BROADCAST,RUNNING,MULTICAST,DHCP,IPv4,PHYSRUNNING> mtu 1500
 index 603
        inet 10.46.86.54 netmask ffffff00 broadcast 10.46.8.255
        ether 0:1e:68:49:98:80
```

Logging Actions of the IPv4 Routing Daemon

If you suspect a malfunction of the IPv4 routing daemon, `routed`, you can start a log that traces the daemon's activity. The log includes all of the packet transfers when you start the `routed` daemon.

Create a log file that traces the routing daemon's actions as follows:

```
# /usr/sbin/in.routed /var/log-file-name
```

Caution - On a busy network, this command can generate almost continuous output.

The following example shows the beginning of the log that is created when you perform the "Logging Actions of the IPv4 Routing Daemon" on page 37 procedure.

```
-- 2003/11/18 16:47:00.000000 --
Tracing actions started
RCVBUF=61440
Add interface lo0  #1   127.0.0.1      -->127.0.0.1/32
<UP|LOOPBACK|RUNNING|MULTICAST|IPv4> <PASSIVE>
Add interface net0 #2   10.10.48.112    -->10.10.48.0/25
<UP|BROADCAST|RUNNING|MULTICAST|IPv4>
turn on RIP
Add    10.0.0.0        -->10.10.48.112      metric=0  net0  <NET_SYN>
Add    10.10.48.85/25  -->10.10.48.112      metric=0  net0  <IF|NOPROP>
```

▼ How to Trace the Activities of the IPv6 Neighbor Discovery Daemon

If you suspect a malfunction of the IPv6 `in.ndpd` daemon, you can start a log that traces the daemon's activity. This trace is displayed on the standard output until it is terminated. This trace includes all packet transfers when you start the `in.ndpd` daemon.

1. **Start a trace of the `in.ndpd` daemon.**

   ```
   # /usr/lib/inet/in.ndpd -t
   ```

2. **Terminate the trace as needed by pressing Control-C.**

Example 7 Tracing the `in.ndpd` Daemon

The following output shows the beginning of a trace of the `in.ndpd` daemon.

```
# /usr/lib/inet/in.ndpd -t
Nov 18 17:27:28 Sending solicitation to  ff02::2 (16 bytes) on net0
Nov 18 17:27:28         Source LLA: len 6 <08:00:20:b9:4c:54>
Nov 18 17:27:28 Received valid advert from fe80::a00:20ff:fee9:2d27 (88 bytes) on net0
Nov 18 17:27:28         Max hop limit: 0
Nov 18 17:27:28         Managed address configuration: Not set
Nov 18 17:27:28         Other configuration flag: Not set
Nov 18 17:27:28         Router lifetime: 1800
Nov 18 17:27:28         Reachable timer: 0
Nov 18 17:27:28         Reachable retrans timer: 0
Nov 18 17:27:28         Source LLA: len 6 <08:00:20:e9:2d:27>
Nov 18 17:27:28         Prefix: 2001:08db:3c4d:1::/64
Nov 18 17:27:28                 On link flag:Set
Nov 18 17:27:28                 Auto addrconf flag:Set
Nov 18 17:27:28                 Valid time: 2592000
Nov 18 17:27:28                 Preferred time: 604800
Nov 18 17:27:28         Prefix: 2002:0a00:3010:2::/64
Nov 18 17:27:28                 On link flag:Set
Nov 18 17:27:28                 Auto addrconf flag:Set
Nov 18 17:27:28                 Valid time: 2592000
Nov 18 17:27:28                 Preferred time: 604800
```

Probing Remote Hosts With the `ping` Command

You can use the `ping` command to determine whether your system can communicate with a remote host. When you run the `ping` command, the ICMP protocol sends a datagram to the host that you specify, asking for a response. ICMP is the protocol that is responsible for error handling on a TCP/IP network. When you use the `ping` command, you can determine whether IP packets can be exchanged between your host and a specified remote host.

The following is the basic syntax of the `ping` command:

```
/usr/sbin/ping host [timeout]
```

where *host* is the name of the remote host. The optional *timeout* argument indicates the time (in seconds) for the `ping` command to continue trying to reach the remote host. The default is 20 seconds. For more information, see the `ping`(1M) man page.

`ping` Command Modifications for IPv6 Support

The `ping` command can use both IPv4 and IPv6 protocols to probe target systems. Protocol selection depends on the addresses that are returned by the name server for the specific target ystem. By default, if the name server returns an IPv6 address for the target system, the `ping` command uses the IPv6 protocol. If the server returns only an IPv4 address, the `ping` command uses the IPv4 protocol. You can override this behavior by using the `-A` option to specify which protocol to use.

For detailed information, see the `ping`(1M) man page.

Determining if a Remote System Is Reachable

To determine if a remote system is reachable, use the `ping` command as follows:

```
% ping hostname
```

If the remote system is accepting ICMP transmissions, the following message is displayed:

```
hostname is alive
```

This message indicates that the remote system responded to the ICMP request. However, if the system is down, cannot receive the ICMP packets, or if there is a routing problem between your system and the remote system, you receive the following response from the `ping` command:

```
no answer from hostname
```

Determining if Packets Between Your System and a Remote System Are Being Dropped

Packet loss can cause network connections to seem sluggish because additional time is spent retransmitting dropped data. Use the `-s` option of the `ping` command to determine if packets between your system and a remote system are being dropped:

```
% ping -s hostname
```

In the following example, the `ping -s` *hostname* command continually sends packets to the specified system until you send an interrupt character or a time out occurs:

```
% ping -s host1.domain8
PING host1.domain8 : 56 data bytes
64 bytes from host1.example.COM (172.16.83.64): icmp_seq=0. time=1.67 ms
64 bytes from host1.example.COM (172.16.83.64): icmp_seq=1. time=1.02 ms
64 bytes from host1.example.COM (172.16.83.64): icmp_seq=2. time=0.986 ms
64 bytes from host1.example.COM (172.16.83.64): icmp_seq=3. time=0.921 ms
64 bytes from host1.example.COM (172.16.83.64): icmp_seq=4. time=1.16 ms
64 bytes from host1.example.COM (172.16.83.64): icmp_seq=5. time=1.00 ms
64 bytes from host1.example.COM (172.16.83.64): icmp_seq=5. time=1.980 ms

^C

----host1.domain8  PING Statistics----
7 packets transmitted, 7 packets received, 0% packet loss
round-trip (ms)  min/avg/max/stddev = 0.921/1.11/1.67/0.26
```

The packet-loss statistic indicates whether the host has any dropped packets. If the `ping` command indicates that there has been packet loss, check the status of the network by using the `ipadm` and `netstat` commands. Refer to "Monitoring IP Interfaces and Addresses" in *Configuring and Managing Network Components in Oracle Solaris 11.3* and "Monitoring Network Status With the `netstat` Command" on page 28 for more details.

Displaying Routing Information With the `traceroute` Command

The `traceroute` command traces the route that an IP packet follows to a remote system. You use the `traceroute` command to uncover any routing misconfiguration and routing path failures. If a particular system is unreachable, you can use `traceroute` to see what path the packet follows to the remote system and where possible failures might be occurring.

The `traceroute` command also displays the round trip time for each gateway along the path to the target system. This information can be useful for analyzing where network traffic is slow between the two systems.

For more details about the `traceroute` command, see the `traceroute`(1M) man page.

This section contains the following topics:

- "`traceroute` Command Modifications for IPv6 Support" on page 41
- "Discovering the Route to a Remote Host" on page 41
- "Tracing All Routes" on page 41

`traceroute` Command Modifications for IPv6 Support

You can use the `traceroute` command to trace both the IPv4 and IPv6 routes to a specific system. From a protocol perspective, the `traceroute` command uses the same algorithm as the `ping` command. Use the `-A` command-line option to override this behavior. Also, you can trace each individual route to every address of a multihomed host by using the -a command-line option.

Discovering the Route to a Remote Host

Use the `traceroute` command as follows to discover the route to a remote system:

```
% traceroute destination-hostname
```

The following output from the `traceroute` command shows the seven–hop path that a packet follows from the local system `nearhost` to the remote system `farhost`. The output also shows the time that it takes a packet to traverse each hop.

```
% traceroute farhost
traceroute to farhost (172.16.64.39), 30 hops max, 40 byte packets
1  frbldg7c-86 (172.16.86.1)  1.516 ms  1.283 ms  1.362 ms
2  bldg1a-001 (172.16.1.211)  2.277 ms  1.773 ms  2.186 ms
3  bldg4-bldg1 (172.16.4.42)  1.978 ms  1.986 ms  13.996 ms
4  bldg6-bldg4 (172.16.4.49)  2.655 ms  3.042 ms  2.344 ms
5  ferbldg11a-001 (172.16.1.236)  2.636 ms  3.432 ms  3.830 ms
6  frbldg12b-153 (172.16.153.72)  3.452 ms  3.146 ms  2.962 ms
7  farhost (172.16.64.39)  3.430 ms  3.312 ms  3.451 ms
```

Tracing All Routes

Use the `traceroute` command with -a option on the local system to trace all routes:

```
% traceroute -a host-name
```

The following example displays all possible routes to a dual-stack host:

```
% traceroute -a v6host
traceroute: Warning: Multiple interfaces found; using 2001:db8:4a3a:1:56:a0:a8 @ net0:2
 traceroute to v6host (2001:db8:4a3b:5:102:a00:fe79:19b0),30 hops max, 60 byte packets
1 v6-rout86 (2001:db8:4a3b:1:56:a00:fe1f:59a1) 35.534 ms 56.998 ms *
2 2001:db8::255:0:c0a8:717 32.659 ms 39.444 ms *
3  farhost (2001:db8:4a3b:2:103:a00:fe9a:ce7b)  401.518 ms  7.143 ms *
4  distant (2001:db8:4a3b:3:100:a00:fe7c:cf35)  113.034 ms  7.949 ms *
5  v6host (2001:db8:4a3b:5:102:a00:fe79:19b0)  66.111 ms *  36.965 ms *
```

```
traceroute to v6host  (192.168.10.75),30 hops max,40 byte packets
1  v6-rout86 (172.16.86.1)  4.360 ms  3.452 ms  3.479 ms
2  flrmpj17u (172.16.17.131)  4.062 ms  3.848 ms  3.505 ms
3  farhost (10.0.0.23)  4.773 ms *  4.294 ms
4  distant (192.168.10.104)  5.128 ms  5.362 ms *
5  v6host  (192.168.15.85)  7.298 ms  5.444 ms *
```

About the My Traceroute Utility

My Traceroute (`mtr`) combines the functionality of the `ping` and `traceroute` commands into a single networking diagnostics tool. The utility sends exploratory packets to a specified system at regular intervals, similar to how the `ping -s` command works. The utility also tracks network hops between the current host and a target system, similar to how the `traceroute` command works. In addition, the utility records and displays timing information on screen. Information that is displayed is updated constantly as new packets are sent out and responses are returned.

To use the `mtr` utility on your Oracle Solaris 11 system, you must first install the `network/mtr` IPS package. Note that the utility uses the same security model that the `traceroute` and `ping` commands use.

For more detailed information, see the `mtr`(1M) man page.

Analyzing Network Traffic With TShark and Wireshark

TShark is a command-line network traffic analyzer that enables you to capture packet data from a live network or read packets from a previously saved *capture file* by either printing a decoded form of those packets to the standard output or by writing the packets to a file. Without any options, TShark works similarly to the `tcpdump` command and also uses the same live capture file format, `libpcap`. In addition, TShark is capable of detecting, reading, and writing the same capture files as those that are supported by Wireshark.

Wireshark is a third-party graphical user interface (GUI) network protocol analyzer that is used to interactively dump and analyze network traffic. Similar to the `snoop` command, you can use Wireshark to browse packet data on a live network or from a previously saved capture file. By default, Wireshark uses the `libpcap` format for file captures, which is also used by the `tcpdump` utility and other similar tools. A key advantage of using Wireshark is that it is capable of reading and importing several other file formats besides the `libpcap` format.

Both TShark and Wireshark provide several unique features, including the following:

- Capable of assembling all of the packets in a TCP conversation and displaying the data in that conversation in ASCII, EBCDIC or hex format
- Contain more filterable fields than in other network protocol analyzers

- Use a syntax that is richer than other network protocol analyzers for creating filters

To use TShark and Wireshark on your Oracle Solaris system, first check that the software packages are installed, and if necessary, install them as follows:

```
# pkg install tshark
```

```
# pkg install wireshark
```

For more information, see the `tshark(1)` and `wireshark(1)` man pages.

See also the Wireshark documentation at `http://www.wireshark.org`.

Monitoring Packet Transfers With the `snoop` Command

You can use the `snoop` command to monitor network traffic. The `snoop` command captures network packets and displays the contents in the format that you specify. Packets can be displayed as soon as they are received or saved to a file. When the `snoop` command writes to an intermediate file, packet loss under busy trace conditions is unlikely. the `snoop` command is then used to interpret the file.

To capture packets to and from the default interface in promiscuous mode, you must assume the Network Management rights profile or the `root` role. In summary form, the `snoop` command displays only the data that pertains to the highest-level protocol. For example, an NFS packet only displays NFS information. The underlying remote procedure call (RPC), UDP, IP, and Ethernet frame information is suppressed but can also be displayed if either of the verbose options is used.

Use the `snoop` command frequently and consistently to become familiar with normal system behavior. For more details about the `snoop` command, refer to the `snoop(1M)` man page.

This section contains the following topics:

`snoop` Command Modifications for IPv6 Support

The `snoop` command can capture both IPv4 and IPv6 packets. This command can display IPv6 headers, IPv6 extension headers, ICMPv6 headers, and Neighbor Discovery (ND) protocol data. By default, the `snoop` command displays both IPv4 and IPv6 packets. If you specify the `ip` or ip6 protocol keyword, then the command displays only IPv4 or IPv6 packets. The IPv6 filter option enables you to filter through all packets, both IPv4 and IPv6, displaying only the IPv6 packets. See "Monitoring IPv6 Network Traffic" on page 46 and the snoop(1M) man page.

▼ How to Check Packets From All Interfaces

1. **Print information about the interfaces that are attached to the system.**

   ```
   # ipadm show-if
   ```

   The snoop command normally uses the first non-loopback device, which is typically the primary network interface.

2. **Begin packet capture by typing snoop without arguments.**

   ```
   # snoop
   ```

3. **Press Control-C to halt the process.**

Example 8 Displaying Basic snoop Output

The following example shows the basic snoop command output for a dual-stack host.

```
# snoop
Using device /dev/net (promiscuous mode)
router5.local.com -> router5.local.com ARP R 10.0.0.13, router5.local.com is
0:10:7b:31:37:80
router5.local.com -> BROADCAST     TFTP Read "network-confg" (octet)
myhost -> DNSserver.local.com      DNS C 192.168.10.10.in-addr.arpa. Internet PTR ?
DNSserver.local.com  foohost        DNS R 192.168.10.10.in-addr.arpa. Internet PTR
niserve2.
.
.
.
fe80::a00:20ff:febb:e09 -> ff02::9 RIPng R (5 destinations)
```

In the previous output, the packets that are captured show a DNS query and response, as well as periodic Address Resolution Protocol (ARP) packets from the local router and advertisements of the IPv6 link-local address to the in.ripngd daemon.

▼ How to Check Packets From an IPMP Group

In Oracle Solaris 10, if you want to capture packets from an IPMP group, you need to manually run the snoop command on each of the interfaces of the IPMP group. In the Oracle Solaris 11 release, you can capture packets from all of the interfaces in an IPMP group by using the `snoop` command with the `-I` option. The output is then consolidated into a single output stream.

The `snoop` command normally uses the first non-loopback device, which is typically the primary network interface. However, when the `-I` option is used, the `snoop` command uses IP interfaces (from `/dev/ipnet/*`), as displayed by the `ipadm show-if` command. You can also use this option to enable the snooping of loopback traffic and other IP-only constructs. The `-d` option uses datalink interfaces, those that are displayed in the output of the `dladm show-link` command.

1. **Print information about the interfaces that are attached to the system.**

```
# ipadm show-if
```

2. **Begin packet capture for the specified IPMP group.**

```
# snoop -I ipmp-group
```

3. **Press Control-C to halt the process.**

▼ How to Capture `snoop` Output to a File

1. **Capture a `snoop` session into a file. For example:**

```
# snoop -o /tmp/cap
Using device /dev/eri (promiscuous mode)
30 snoop: 30 packets captured
```

In the previous example, 30 packets have been captured in a file named `/tmp/cap`. The file can be in any directory that has enough disk space. The number of packets that are captured is displayed on the command line, enabling you to press Control-C to abort at any time.

The `snoop` command creates a noticeable network load on the host system, which can distort the results. To see the actual results, run the `snoop` command from a third system.

2. **Inspect the `snoop` output capture file.**

```
# snoop -i filename
```

Example 9 Displaying `snoop` Output Captures

The following output shows a capture that you might receive as output from the `snoop -i` command.

```
# snoop -i /tmp/cap
1   0.00000 fe80::a00:20ff:fee9:2d27 -> fe80::a00:20ff:fecd:4375
```

```
ICMPv6 Neighbor advertisement
...
10  0.91493    10.0.0.40 -> (broadcast)  ARP C Who is 10.0.0.40, 10.0.0.40 ?
34  0.43690 nearserver.example.com  -> 224.0.1.1  IP  D=224.0.1.1 S=10.0.0.40 LEN=28,
ID=47453, TO =0x0, TTL=1
35  0.00034  10.0.0.40 -> 224.0.1.1    IP  D=224.0.1.1 S=10.0.0.40 LEN=28, ID=57376,
TOS=0x0, TTL=47
```

▼ How to Check Packets Between an IPv4 Server and a Client

1. **Establish a `snoop` system off a hub (or a mirrored port on a switch) that is connected to either the IPv4 client or the IPv4 server.**

   The third system (the `snoop` system) checks all of the intervening traffic, so that the `snoop` trace reflects what is actually happening on the wire.

2. **Type the `snoop` command with the appropriate options and save the output to a file.**

3. **Inspect and interpret the output.**

   Refer to RFC 1761, Snoop Version 2 Packet Capture File Format (`http://www.rfc-editor.org/rfc/rfc1761.txt`) for details of the `snoop` capture file.

Monitoring IPv6 Network Traffic

Use the `snoop` command as follows to capture IPv6 packets:

```
# snoop ip6
```

The following example shows typical output that might be displayed when you run the `snoop ip6` command on a IPv6 node.

```
# snoop ip6
fe80::a00:20ff:fecd:4374 -> ff02::1:ffe9:2d27 ICMPv6 Neighbor solicitation
fe80::a00:20ff:fee9:2d27 -> fe80::a00:20ff:fecd:4375 ICMPv6 Neighbor
solicitation
fe80::a00:20ff:fee9:2d27 -> fe80::a00:20ff:fecd:4375 ICMPv6 Neighbor
solicitation
fe80::a00:20ff:febb:e09 -> ff02::9      RIPng R (11 destinations)
fe80::a00:20ff:fee9:2d27 -> ff02::1:ffcd:4375 ICMPv6 Neighbor solicitation
```

For more information on the `snoop` command, see the `snoop`(1M) man page.

Monitoring Packets by Using IP Layer Devices

IP layer devices are introduced in Oracle Solaris to enhance IP observability. These devices provide access to all of the packets with addresses that are associated with the system's network interface. The addresses include local addresses as well as addresses that are hosted on non-loopback interfaces or logical interfaces. The observable traffic can be both IPv4 and IPv6 addresses. Thus, you can monitor all traffic that is destined for the system. The traffic can be loopback IP traffic, packets from remote machines, packets that are being sent from the system, or all forwarded traffic.

With IP layer devices, an administrator for an Oracle Solaris global zone can monitor traffic between zones, as well as within a zone. An administrator of a non-global zone can also observe traffic that is sent and received by that zone.

To monitor traffic on the IP layer, use the `snoop` command with the newer `-I`. This option specifies that the command use the new IP layer devices instead of the underlying link-layer device to display traffic data.

▼ How to Check Packets on the IP Layer

1. **(Optional) If necessary, print the information about the interfaces that are attached to the system.**

```
# ipadm show-if
```

2. **Capture IP traffic on a specific interface.**

```
# snoop -I interface [-V | -v]
```

Methods for Checking Packets

All of the following examples are based on this system configuration:

```
# ipadm show-addr

ADDROBJ           TYPE     STATE        ADDR
lo0/v4            static   ok           127.0.0.1/8
net0/v4           dhcp     ok           10.153.123.225/24
lo0/v6            static   ok           ::1/128
net0/v6           addrconf ok           fe80::214:4fff:2731:b1a9/10
net0/v6           addrconf ok           2001:0db8:212:60bb:214:4fff:2731:b1a9/64
net0/v6           addrconf ok           2001:0db8:56::214:4fff:2731:b1a9/64
```

Suppose that two zones, `sandbox` and `toybox`, are using the following IP addresses:

- `sandbox` – `172.0.0.3`

- toybox – 172.0.0.1

You can use the snoop -I command on the different interfaces that are on the system. The packet information that is displayed depends on whether you are an administrator for the global zone or for the non-global zone.

EXAMPLE 10 Observing Traffic on the Loopback Interface

The following example shows the output of the snoop command for the loopback interface.

```
# snoop -I lo0
Using device ipnet/lo0 (promiscuous mode)
localhost -> localhost    ICMP Echo request (ID: 5550 Sequence number: 0)
localhost -> localhost    ICMP Echo reply (ID: 5550 Sequence number: 0)
```

To generate verbose output, use the -v option.

```
# snoop -v -I lo0
Using device ipnet/lo0 (promiscuous mode)
IPNET:  ----- IPNET Header -----
IPNET:
IPNET:  Packet 1 arrived at 10:40:33.68506
IPNET:  Packet size = 108 bytes
IPNET:  dli_version = 1
IPNET:  dli_type = 4
IPNET:  dli_srczone = 0
IPNET:  dli_dstzone = 0
IPNET:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
...
```

Support for observing packets on the IP layer introduces a new ipnet header that precedes the packets that are being observed. Both the source and destination IDs are indicated. The 0 ID indicates that the traffic is being generated from the global zone.

EXAMPLE 11 Observing Packet Flow in the net0 Device Within Local Zones

The following example shows traffic that occurs in the different zones that are within the system. You can see all of the packets that are associated with the net0 IP addresses, including those packets that are locally delivered to other zones. If you generate verbose output, you can also see the zones that are involved in the flow of packets.

```
# snoop -I net0
Using device ipnet/net0 (promiscuous mode)
toybox -> sandbox TCP D=22 S=62117 Syn Seq=195630514 Len=0 Win=49152 Options=<mss
sandbox -> toybox TCP D=62117 S=22 Syn Ack=195630515 Seq=195794440 Len=0 Win=49152
```

```
toybox -> sandbox TCP D=22 S=62117 Ack=195794441 Seq=195630515 Len=0 Win=49152
sandbox -> toybox TCP D=62117 S=22 Push Ack=195630515 Seq=195794441 Len=20 Win=491
```

```
# snoop -I net0 -v port 22
IPNET:  ----- IPNET Header -----
IPNET:
IPNET:  Packet 5 arrived at 15:16:50.85262
IPNET:  Packet size = 64 bytes
IPNET:  dli_version = 1
IPNET:  dli_type = 0
IPNET:  dli_srczone = 0
IPNET:  dli_dstzone = 1
IPNET:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:         .... ..0. = not ECN capable transport
IP:         .... ...0 = no ECN congestion experienced
IP:   Total length = 40 bytes
IP:   Identification = 22629
IP:   Flags = 0x4
IP:         .1.. .... = do not fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 64 seconds/hops
IP:   Protocol = 6 (TCP)
IP:   Header checksum = 0000
IP:   Source address = 172.0.0.1, 172.0.0.1
IP:   Destination address = 172.0.0.3, 172.0.0.3
IP:   No options
IP:
TCP:  ----- TCP Header -----
TCP:
TCP:  Source port = 46919
TCP:  Destination port = 22
TCP:  Sequence number = 3295338550
TCP:  Acknowledgement number = 3295417957
TCP:  Data offset = 20 bytes
TCP:  Flags = 0x10
TCP:        0... .... = No ECN congestion window reduced
TCP:        .0.. .... = No ECN echo
TCP:        ..0. .... = No urgent pointer
TCP:        ...1 .... = Acknowledgement
TCP         .... 0... = No push
TCP         .... .0.. = No reset
TCP:        .... ..0. = No Syn
```

```
TCP:        .... ...0 = No Fin
TCP:  Window = 49152
TCP:  Checksum = 0x0014
TCP:  Urgent pointer = 0
TCP:  No options
TCP:
```

In the previous output, the `ipnet` header indicates that the packet is coming from the global zone (ID `0`) to `sandbox` (ID `1`).

EXAMPLE 12 Observing Network Traffic by Identifying a Zone

The following example shows how to observe network traffic by identifying the zone, which is extremely useful on systems that have multiple zones. Currently, you can only identify by zone by using the zone ID. Using the `snoop` command with zone names is not supported.

```
# snoop -I hme0 sandboxsnoop -I net0 sandbox
Using device ipnet/hme0 (promiscuous mode)
toybox -> sandbox TCP D=22 S=61658 Syn Seq=374055417 Len=0 Win=49152 Options=<mss
sandbox -> toybox TCP D=61658 S=22 Syn Ack=374055418 Seq=374124525 Len=0 Win=49152
toybox -> sandbox TCP D=22 S=61658 Ack=374124526 Seq=374055418 Len=0 Win=49152
```

Observing Network Traffic With the `ipstat` and `tcpstat` Commands

Two new commands for observing various types of network traffic on a server are introduced in this release: `ipstat` and `tcpstat`.

The `ipstat` command is used to gather and report statistics about IP traffic on a server based on the selected output mode and sort order that is specified in the command syntax. This command enables you to observe network traffic at the IP layer, aggregated on source, destination, higher-layer protocol, and interface. Use this command when you want to observe the amount of traffic between one server and other servers.

The `tcpstat` command is used to gather and report statistics on TCP and UDP traffic on a server based on the selected output mode and sort order that is specified in the command syntax. This command enables you to observe network traffic at the transport layer, specifically for TCP and UDP. In addition to the source and destination IP addresses, you can observe the source and destination TCP or UDP ports, the PID of the process that is sending or receiving the traffic, and the name of the zone in which that process is running.

The following are some of the ways in which you can use the `tcpstat` command:

- Identify the largest sources of TCP and UDP traffic on a server.
- Examine the traffic that is being generated by a particular process.

- Examine the traffic that is being generated from a particular zone.
- Determine which process is bound to a local port.

Note - The previous list is not exhaustive. There are several other ways in which you can use the `tcpstat` command. See the `tcpstat`(1M) man page for more information.

To use the `ipstat` and `tcpstat` commands, one of the following privileges is required:

- Assume the `root` role
- Be explicitly assigned the `dtrace_kernel` privilege
- Be assigned either the Network Management or the Network Observability rights profile

The following examples show various ways in which you can use these two commands to observe network traffic. For detailed information, see the `tcpstat`(1M) and `ipstat`(1M) man pages.

The following example shows output from the `ipstat` command when run with the `-c` option. Use the `-c` option to print newer reports after previous reports, without overwriting the previous report. The number 3 in this example specifies the interval for displaying data, which is the same as if the command were invoked as `ipstat 3`.

```
# ipstat -c 3
SOURCE                     DEST                       PROTO    INT        BYTES
zucchini                   antares                    TCP      net0       72.0
zucchini                   antares                    SCTP     net0       64.0
antares                    zucchini                   SCTP     net0       56.0
amadeus.foo.example.com    10.6.54.255                UDP      net0       40.0
antares                    zucchini                   TCP      net0       40.0
zucchini                   antares                    UDP      net0       16.0
antares                    zucchini                   UDP      net0       16.0
Total: bytes in: 192.0  bytes out: 112.0
```

By comparison, the following example shows output of the `tcpstat` command when used with the `-c` option:

```
# tcpstat -c 3
ZONE         PID PROTO  SADDR             SPORT DADDR             DPORT   BYTES
global    100680 UDP    antares           62763 agamemnon          1023   76.0
global    100680 UDP    antares             775 agamemnon          1023   38.0
global    100680 UDP    antares             776 agamemnon          1023   37.0
global    100680 UDP    agamemnon          1023 antares           62763   26.0
global    104289 UDP    zucchini          48655 antares            6767   16.0
global    104289 UDP    clytemnestra      51823 antares            6767   16.0
global    104289 UDP    antares            6767 zucchini          48655   16.0
global    104289 UDP    antares            6767 clytemnestra      51823   16.0
global    100680 UDP    agamemnon          1023 antares             776   13.0
global    100680 UDP    agamemnon          1023 antares             775   13.0
global    104288 TCP    zucchini          33547 antares            6868    8.0
global    104288 TCP    clytemnestra      49601 antares           6868    8.0
global    104288 TCP    antares            6868 zucchini         33547    8.0
global    104288 TCP    antares            6868 clytemnestra     49601    8.0
Total: bytes in: 101.0  bytes out: 200.0
```

The following additional examples show other ways in which you can observe traffic on your network by using the `ipstat` and `tcpstat` commands.

EXAMPLE 13 Observing the Five Most Active IP Traffic Flows by Using the `ipstat` Command

The following example reports the five most active IP traffic flows. The `-l` *nlines* option specifies how many lines of data to output per report.

```
# ipstat -l 5
SOURCE                     DEST                       PROTO    IFNAME     BYTES
charybdis.foo.example.com  achilles.exampl            UDP      net0        6.6K
eratosthenes.example.com   aeneas.example.c           TCP      tun0        6.1K
achilles.exampl            charybdis.foo.example.com  UDP      net0      964.0
aeneas.example.c           eratosthenes.example.com   TCP      tun0      563.0
odysseus.example.          255.255.255.255            UDP      net0       66.0
Total: bytes in: 12.6K bytes out:  2.2K
```

EXAMPLE 14 Displaying a Time Stamp by Using the `ipstat` Command

The following example reports the top IP traffic with a time stamp in standard date format (`-d` d). You can specify that the timestamp be printed in seconds, or Unix time (`-d` u). The interval is set to 10 seconds.

```
# ipstat -d d -c 10
Monday, March 26, 2012 08:34:07 PM EDT
SOURCE                     DEST                       PROTO    IFNAME     BYTES
charybdis.foo.example.com  achilles.exampl            UDP      net0       15.1K
eratosthenes.example.com   aeneas.example.c           TCP      tun0       13.9K
achilles.exampl            charybdis.foo.example.com  UDP      net0        2.4K
aeneas.example.c           eratosthenes.example.com   TCP      tun0        1.5K
odysseus.example.          255.255.255.255            UDP      net0       66.0
cassiopeia.foo.example.com aeneas.example.c           TCP      tun0       29.0
aeneas.example.c           cassiopeia.foo.example.com TCP      tun0       20.0
Total: bytes in: 29.1K bytes out:  3.8K
```

EXAMPLE 15 Observing the Five Most Active Traffic Flows by Using the `tcpstat` Command

The following example reports the five most active TCP traffic flows for a server:

```
# tcpstat -l 5
ZONE            PID PROTO  SADDR            SPORT DADDR            DPORT   BYTES
global        28919 TCP    achilles.exampl  65398 aristotle.exampl   443   33.0
zone1          6940 TCP    ajax.example.com  6868 achilles.exampl  61318    8.0
zone1          6940 TCP    achilles.exampl  61318 ajax.example.com  6868    8.0
```

```
global         8350 TCP    ajax.example.com  6868 achilles.exampl  61318    8.0
global         8350 TCP    achilles.exampl  61318 ajax.example.com  6868    8.0
Total: bytes in: 16.0  bytes out: 49.0
```

EXAMPLE 16 Displaying Timestamp Information by Using the `tcpstat` Command

In the following example, the `tcpstat` command is used to display timestamp information for TCP network traffic on a server in standard date format:

```
# tcpstat -d d -c 10
Saturday, March 31, 2012 07:48:05 AM EDT
ZONE            PID PROTO  SADDR            SPORT DADDR            DPORT   BYTES
global         2372 TCP    penelope.example 58094 polyphemus.examp    80   37.0
zone1          6940 TCP    ajax.example.com  6868 achilles.exampl  61318    8.0
zone1          6940 TCP    achilles.exampl  61318 ajax.example.com  6868    8.0
global         8350 TCP    ajax.example.com  6868 achilles.exampl  61318    8.0
global         8350 TCP    achilles.exampl  61318 ajax.example.com  6868    8.0
Total: bytes in: 16.0  bytes out: 53.0
```

♦ ♦ ♦ C H A P T E R 2

About IPMP Administration

IP network multipathing (IPMP) is a Layer 3 (L3) technology that enables you to group multiple IP interfaces into a single logical interface. With features such as failure detection, transparent access failover, and packet load spreading, IPMP improves network performance by ensuring that the network is always available to the system.

This chapter contains the following topics:

- "What's New in IPMP" on page 55
- "IPMP Support in Oracle Solaris" on page 57
- "IPMP Addressing" on page 67
- "Failure Detection in IPMP" on page 68
- "Detecting Physical Interface Repairs" on page 71
- "IPMP and Dynamic Reconfiguration" on page 73

Note - Throughout this chapter and in Chapter 3, "Administering IPMP", all references to the term *interface* specifically means *IP interface*. Unless a qualification explicitly indicates a different use of the term, such as a network interface card (NIC), the term always refers to the interface that is configured on the IP layer.

What's New in IPMP

The following features are new or have changed in this release.

Outbound Load Spreading Improvements

Outbound load spreading for IPMP has been enhanced to occur on a per-connection basis, rather than on a next-hop basis as in previous Oracle Solaris releases. This improved functionality works by enabling two different connections to the same off-link destination using different outbound interfaces. See "IPMP Support in Oracle Solaris" on page 57.

IPMP Configuration Changes

IPMP works differently in this release than in Oracle Solaris 10. One significant change is that IP interfaces are now grouped into a *virtual* IP interface, for example, `ipmp0`. The virtual IP interface serves all of the data IP addresses, while test addresses that are used for probe-based failure detection are assigned to an underlying interface such as `net0`. For more information, see "How IPMP Works" on page 61.

Refer to the following general work flow when transitioning from your existing IPMP configuration to the new IPMP model:

1. Make sure that the `DefaultFixed` profile is enabled on your system prior to configuring IPMP. See "Enabling and Disabling Profiles" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.
2. Ensure that MAC addresses on SPARC based systems are unique. See "How to Ensure That the MAC Address of Each Interface Is Unique" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.
3. Use the `dladm` command to configure datalinks. To use the same physical network devices within your IPMP configuration, you will need to first identify the datalinks that are associated with each device instance:

```
# dladm show-phys
LINK            MEDIA               STATE      SPEED  DUPLEX    DEVICE
net1            Ethernet            unknown    0      unknown   bge1
net0            Ethernet            up         1000   full      bge0
net2            Ethernet            unknown    1000   full      e1000g0
net3            Ethernet            unknown    1000   full      e1000g1
```

4. If you previously used `e1000g0` and `e1000g1` for your IPMP configuration, you would now use `net2` and `net3`. Note that datalinks can be based not only on physical links but also on aggregations, VLANs, VNICs, and so on. For more information, see "Displaying a System's Datalinks" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.
5. Use the `ipadm` command to perform the following tasks:
   - Configure the network layer.
   - Create IP interfaces.
   - Add IP interfaces to an IPMP group.
   - Add data addresses to an IPMP group.

For more information about network administration command changes in this release, see "Network Administration Command Changes" in *Transitioning From Oracle Solaris 10 to Oracle Solaris 11.3*.

IPMP Support in Oracle Solaris

IPMP provides the following support:

- IPMP enables you to configure multiple IP interfaces into a single group, called an IPMP group. As a whole, the IPMP group with its multiple underlying IP interfaces is represented as a single *IPMP interface*. This interface is treated just like any other interface on the IP layer of the network stack. All IP administrative tasks, routing tables, Address Resolution Protocol (ARP) tables, firewall rules, and other IP-related procedures work with an IPMP group by referring to the IPMP interface.

  Note - Although Oracle Solaris supports the use of iSCSI devices with IPMP, a server that boots from an iSCSI device cannot be part of an IPMP group.

- The system handles the distribution of data addresses amongst the underlying active interfaces. When the IPMP group is created, data addresses belong to the IPMP interface as an *address pool*. The kernel then automatically and randomly binds the data addresses to the underlying active interfaces of the group.
- You primarily use the `ipmpstat` command to obtain information about IPMP groups. This command provides information about all aspects of the IPMP configuration, such as the underlying IP interfaces of the group, test and data addresses, types of failure detection being used, and which interfaces have failed. The `ipmpstat` command functions, the options that you can use, and the output that each option generates are all described in "Monitoring IPMP Information" on page 94.
- You can assign an IPMP interface a customized name to identify the IPMP group more easily. See "Configuring IPMP Groups" on page 75.

Benefits of Using IPMP

Different factors can cause an interface to become unusable, such as an interface failure or interfaces being taken offline for maintenance. Without IPMP, the system can no longer be contacted by using any of the IP addresses that are associated with that unusable interface. Additionally, existing connections that use those IP addresses are disrupted.

With IPMP, you can configure multiple IP interfaces into an *IPMP group*. The group functions like an IP interface with data addresses to send or receive network traffic. If an underlying interface in the group fails, the data addresses are redistributed amongst the remaining underlying active interfaces in the group. Thus, the group maintains network connectivity despite an interface failure. With IPMP, network connectivity is always available, provided that a minimum of one interface is usable for the group.

IPMP also improves overall network performance by automatically spreading outbound network traffic across the set of interfaces within the IPMP group. This process is called *outbound load spreading*. The system also indirectly controls inbound load spreading by performing source address selection for packets whose IP source address was not specified by the application. However, if an application has explicitly chosen an IP source address, then the system does not vary that source address.

In this release, outbound load spreading occurs on a per-connection basis, rather than on a next hop basis as in previous releases. This change greatly improves IPMP capabilities by enabling two different connections to the same off-link destination by using different outbound interfaces.

Link aggregations perform functions that are similar to IPMP to improve network performance and availability. To compare these two technologies, see Appendix A, "Link Aggregations and IPMP: Feature Comparison," in *Managing Network Datalinks in Oracle Solaris 11.3*.

Rules for Using IPMP

IPMP group configuration is determined by your specific system configuration.

Observe the following rules for IPMP configuration:

1. Multiple IP interfaces that are on the same LAN must be configured into an IPMP group. A LAN broadly refers to a variety of local network configurations, including VLANs and both wired and wireless local networks with nodes that belong to the same link-layer broadcast domain.

   Note - Multiple IPMP groups on the same link layer (L2) broadcast domain are unsupported. An L2 broadcast domain typically maps to a specific subnet. Therefore, you must configure only one IPMP group per subnet. Note also that some exceptions to this rule apply, for example, in the case of certain engineered systems that are provided by Oracle. For further clarification, contact your Oracle support representative.

2. Underlying IP interfaces of an IPMP group must not span different LANs.

   For example, suppose that a system with three interfaces is connected to two separate LANs. Two IP interfaces connect to one LAN while a single IP interface connects to the other LAN. In this case, the two IP interfaces connecting to the first LAN must be configured as an IPMP group, as required by the first rule. In compliance with the second rule, the single IP interface that connects to the second LAN cannot become a member of that IPMP group. No IPMP configuration is required for the single IP interface. However, you can configure the single interface into an IPMP group to monitor the interface's availability. Single-interface IPMP configuration is discussed further in "Types of IPMP Interface Configurations" on page 60.

   Consider another case where the link to the first LAN consists of three IP interfaces while the other link consists of two interfaces. This setup requires the configuration of two IPMP groups: a three-interface group that connects to the first LAN, and a two-interface group that connects to the second LAN.

3. All interfaces in the same group must have the same STREAMS modules configured in the same order. When planning an IPMP group, first check the order of STREAMS modules on all interfaces in the prospective IPMP group, then push the modules of each interface in the standard order for the IPMP group. To print a list of STREAMS modules, use the `ifconfig` *interface* `modlist` command. For example, here is the `ifconfig` output for a `net0` interface:

   ```
   # ifconfig net0 modlist
   0 arp
   1 ip
   2 e1000g
   ```

   As the previous output shows, interfaces normally exist as network drivers directly below the IP module. These interfaces do not require additional configuration. However, certain technologies are pushed as STREAMS modules between the IP module and the network driver. If a STREAMS module is stateful, then unexpected behavior can occur on failover, even if you push the same module to all of the interfaces in a group. However, you can use stateless STREAMS modules, provided that you push them in the same order on all interfaces in the IPMP group.

   The following example shows the command you might use to use to push the modules of each interface in the standard order for the IPMP group:

   ```
   # ifconfig net0 modinsert vpnmod@3
   ```

   For step-by-step instructions on planning an IPMP group, see "How to Plan an IPMP Group" on page 76.

IPMP Components

The following are the IPMP software components:

- **Multipathing daemon** (`in.mpathd`) - Detects interface failures and repairs. The daemon performs both link-based failure detection and probe-based failure detection if test addresses are configured for the underlying interfaces. Depending on the type of failure detection method that is used, the daemon sets or clears the appropriate flags on the interface to indicate whether the interface failed or has been repaired. As an option, you can also configure the daemon to monitor the availability of all interfaces, including interfaces that are not configured to belong to an IPMP group. For a description of failure detection, see "Failure Detection in IPMP" on page 68.

  The `in.mpathd` daemon also controls the designation of active interfaces in the IPMP group. The daemon attempts to maintain the same number of active interfaces that was originally configured when the IPMP group was created. Thus, `in.mpathd` activates or deactivates underlying interfaces as needed to be consistent with the administrator's configured policy. For more information about how the `in.mpathd` daemon manages the activation of underlying interfaces, see "How IPMP Works" on page 61. For more information about the daemon, refer to the `in.mpathd`(1M) man page.

- **IP kernel module** - Manages outbound load spreading by distributing the set of available IP data addresses in the IPMP group across the set of available underlying IP interfaces in the group. The module also performs source address selection to manage inbound load spreading. Both roles of the module improve network traffic performance.
- **IPMP configuration file** (`/etc/default/mpathd`) - Defines the daemon's behavior.

  You customize the file to set the following parameters:

  - Target interfaces to probe when running probe-based failure detection
  - Time duration to probe a target to detect failure
  - Status with which to flag a failed interface after that interface is repaired
  - Scope of IP interfaces to monitor, whether to also include IP interfaces in the system that are not configured to belong to IPMP groups

  For information about how to modify the configuration file, see "How to Configure the Behavior of the IPMP Daemon" on page 92.
- `ipmpstat` **command** - Provides different types of information about the status of IPMP as a whole. The tool also displays other information about the underlying IP interfaces for each IPMP group, as well as data and test addresses that have been configured for the group. For more information about this command, see "Monitoring IPMP Information" on page 94 and the `ipmpstat`(1M) man page.

Types of IPMP Interface Configurations

An IPMP configuration typically consists of two or more physical interfaces on the same system that are attached to the same LAN.

These interfaces can belong to an IPMP group in either of the following configurations:

- **Active-active configuration** – An IPMP group in which all underlying interfaces are active. An *active interface* is an IP interface that is currently available for use by the IPMP group.

  Note - By default, an underlying interface becomes active when you configure the interface to become part of an IPMP group.

- **Active-standby configuration** – An IPMP group in which at least one interface is administratively configured as a *standby interface*. Although idle, the standby interface is monitored by the multipathing daemon to track the interface's availability, depending on how the interface is configured. If link-failure notification is supported by the interface, link-based failure detection is used. If the interface is configured with a test address, probe-based failure detection is also used. If an active interface fails, the standby interface is automatically deployed as needed. You can configure as many standby interfaces as are needed for an IPMP group.

You can also configure a single interface as its own IPMP group. The single-interface IPMP group behaves the same as an IPMP group with multiple interfaces. However, this IPMP configuration does not provide high availability for network traffic. If the underlying interface fails, then the system loses all capability to send or receive traffic. The purpose of configuring a single-interface IPMP group is to monitor the availability of the interface by using failure detection. By configuring a test address on the interface, the multipathing daemon can track the interface by using probe-based failure detection.

Typically, a single-interface IPMP group configuration is used with other technologies that have broader failover capabilities, such as the Oracle Solaris Cluster software. The system can continue to monitor the status of the underlying interface, but the Oracle Solaris Cluster software provides the functionality to ensure availability of the network when a failure occurs. For more information about the Oracle Solaris Cluster software, see *Oracle Solaris Cluster 4.3 Concepts Guide*.

An IPMP group without underlying interfaces can also exist, for example, a group with underlying interfaces that have been removed. The IPMP group is not destroyed, but the group can no longer be used to send and receive traffic. As underlying interfaces are brought online for the group, then the data addresses of the IPMP interface are allocated to these interfaces and the system resumes hosting network traffic.

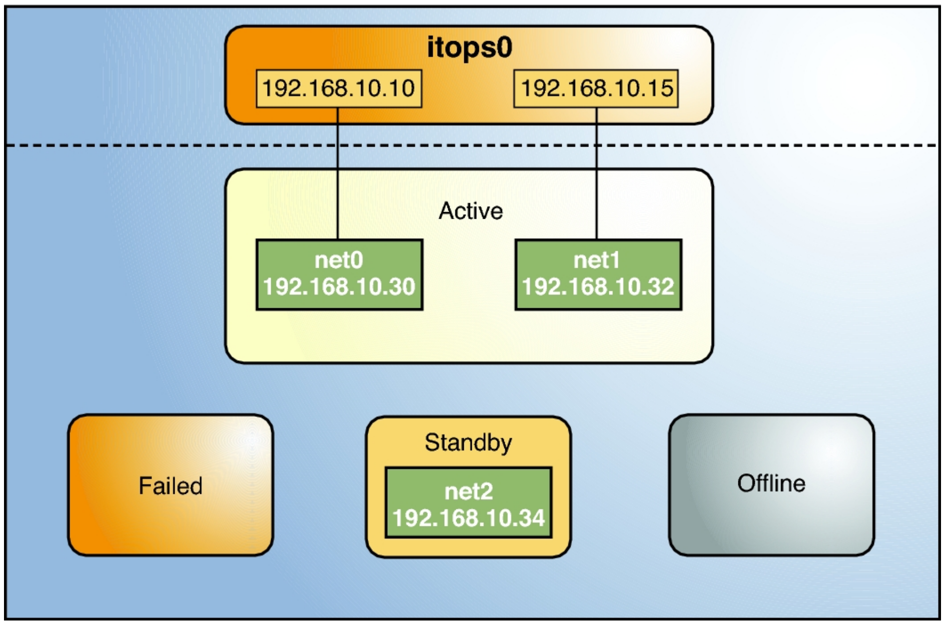
How IPMP Works

IPMP maintains network availability by attempting to preserve the same number of active and standby interfaces that was originally configured when the IPMP group was created.

IPMP failure detection can be link-based, probe-based, or both to determine the availability of a specific underlying IP interface in the group. If IPMP determines that an underlying interface has failed, then that interface is flagged as failed and is no longer usable. The data IP address that was associated with the failed interface is then redistributed to another functioning interface in the group. If available, a standby interface is also deployed to maintain the original number of active interfaces.

Consider a three-interface IPMP group, `itops0`, with an active-standby configuration, as illustrated in the following figure.

FIGURE 1 IPMP Active-Standby Configuration



The IPMP group `itops0` is configured as follows:

- Two data addresses are assigned to the group: `192.168.10.10` and `192.168.10.15`.
- Two underlying interfaces are configured as active interfaces and are assigned flexible link names: `net0` and `net1`.
- The group has one standby interface, also with a flexible link name: `net2`.
- Probe-based failure detection is used, and thus the active and standby interfaces are configured with test addresses as follows:
  - `net0`: `192.168.10.30`
  - `net1`: `192.168.10.32`
  - `net2`: `192.168.10.34`

Note - The Active, Offline, Standby, and Failed areas in Figure 1, "IPMP Active-Standby Configuration," on page 62, Figure 2, "Interface Failure in IPMP," on page 64, Figure 3, "Standby Interface Failure in IPMP," on page 65, and Figure 4, "IPMP Recovery Process," on page 66 indicate only the status of underlying interfaces and not the physical locations. No physical movement of interfaces or addresses, or any transfer of IP interfaces, occurs within this IPMP implementation. The areas only serve to show how an underlying interface changes status as a result of either failure or repair.

You can use the `ipmpstat` command with different options to display specific types of information about existing IPMP groups. For additional examples, see "Monitoring IPMP Information" on page 94.

The following command displays information about the IPMP configuration in Figure 1, "IPMP Active-Standby Configuration," on page 62:

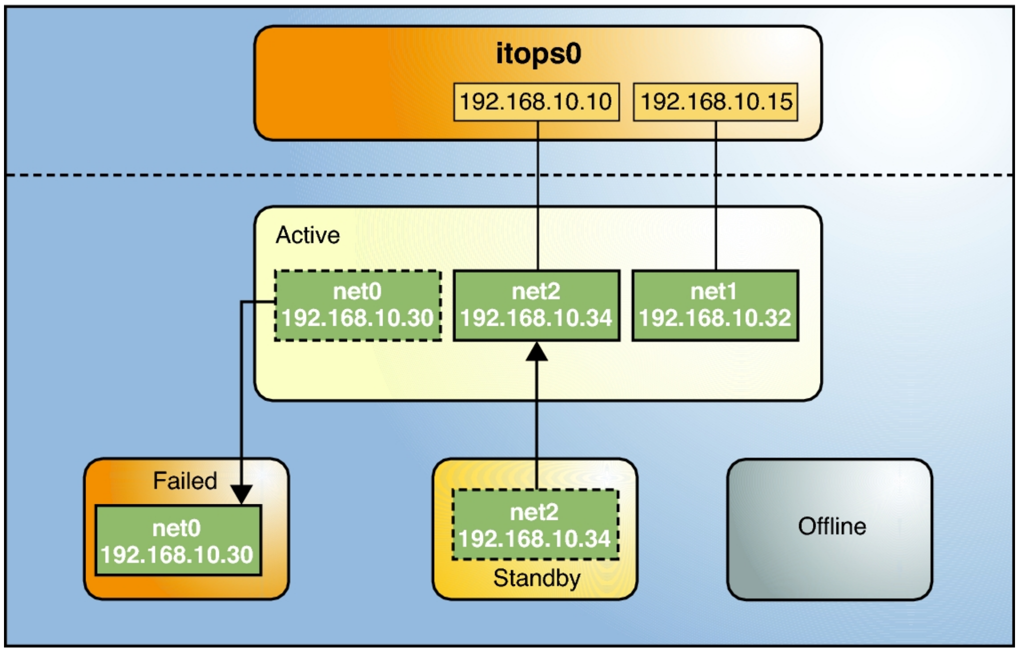
```
# ipmpstat -g
GROUP     GROUPNAME     STATE     FDT        INTERFACES
itops0    itops0        ok        10.00s     net1 net0 (net2)
```

You would display information about the group's underlying interfaces as follows:

```
# ipmpstat -i
INTERFACE   ACTIVE     GROUP     FLAGS      LINK        PROBE     STATE
net0        yes        itops0    -------    up          ok        ok
net1        yes        itops0    --mb---    up          ok        ok
net2        no         itops0    is-----    up          ok        ok
```

IPMP maintains network availability by managing the underlying interfaces to preserve the original number of active interfaces. Thus, if `net0` fails, then `net2` is deployed to ensure that the IPMP group continues to have two active interfaces. The `net2` activation is shown in the following figure.

FIGURE 2 Interface Failure in IPMP



Note - The one-to-one mapping of data addresses to active interfaces in Figure 2, "Interface Failure in IPMP," on page 64 serves only to simplify the illustration. The IP kernel module can randomly assign data addresses without necessarily adhering to a one-to-one relationship between data addresses and interfaces.

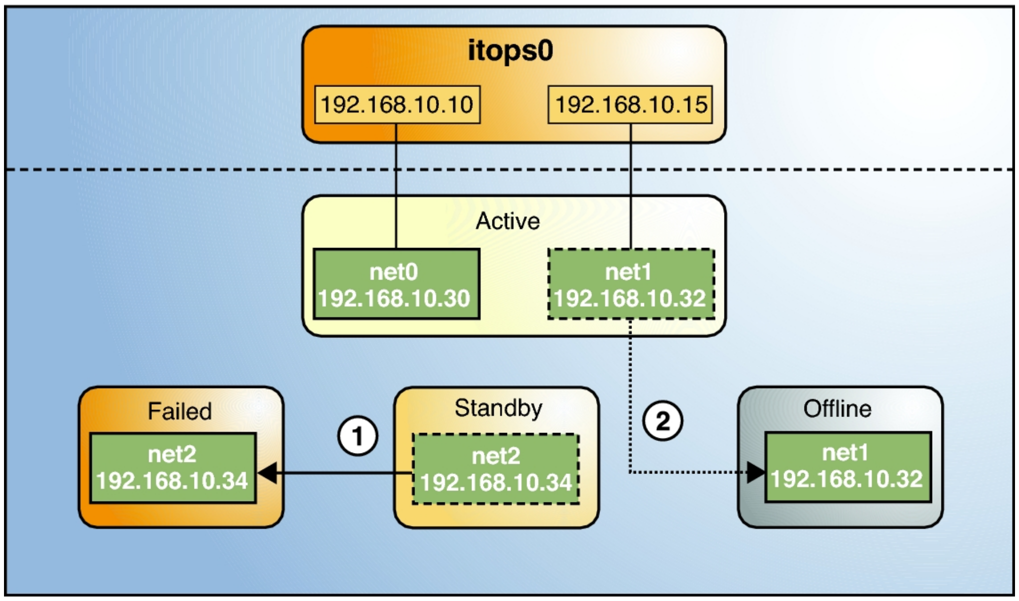
The `ipmpstat` command displays the information in the figure as follows:

```
# ipmpstat -i
INTERFACE   ACTIVE     GROUP     FLAGS      LINK        PROBE     STATE
net0        no         itops0    -------    up          failed    failed
net1        yes        itops0    --mb---    up          ok        ok
net2        yes        itops0    -s-----    up          ok        ok
```

After `net0` is repaired, it reverts to its status as an active interface. In turn, `net2` is returned to its original standby status.

A different failure scenario is shown in Figure 3, "Standby Interface Failure in IPMP," on page 65, where the standby interface `net2` fails (1). Later, one active interface, `net1`, is taken offline by the administrator (2). The result is that the IPMP group is left with a single functioning interface, `net0`.

FIGURE 3 Standby Interface Failure in IPMP

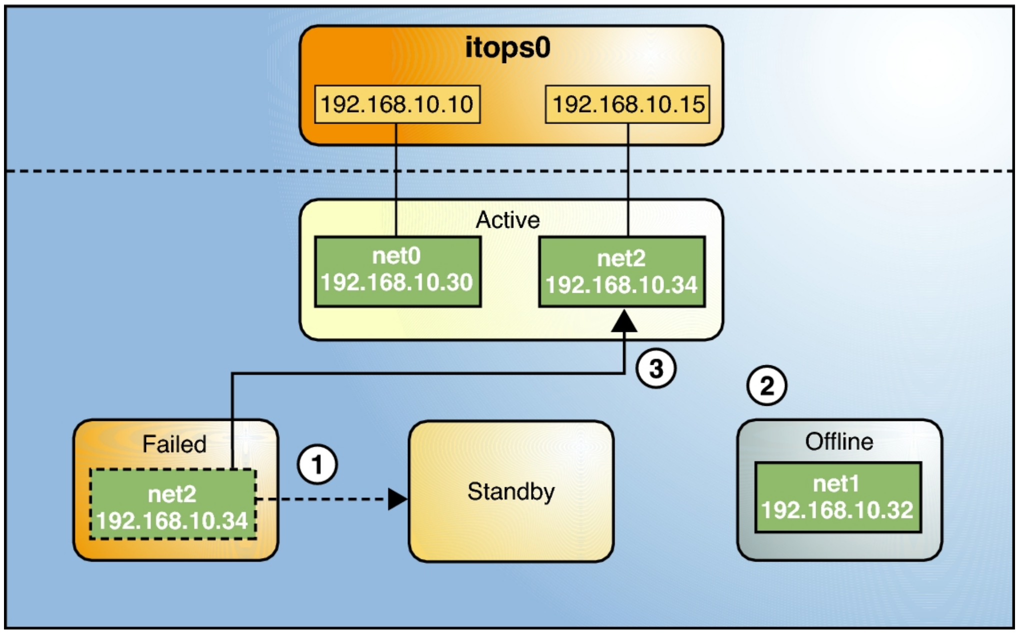


The `ipmpstat` command displays the information in the figure as follows:

```
# ipmpstat -i
INTERFACE   ACTIVE     GROUP     FLAGS      LINK       PROBE     STATE
net0        yes        itops0    -------    up         ok        ok
net1        no         itops0    --mb-d-    up         ok        offline
net2        no         itops0    is-----    up         failed    failed
```

For this particular failure, the recovery process after an interface is repaired is different. The recovery process depends on the IPMP group's original number of active interfaces compared with the configuration after the repair. The following figure represents the recovery process.

FIGURE 4 IPMP Recovery Process



In Figure 4, “IPMP Recovery Process,” on page 66, when `net2` is repaired, it would normally revert to its original status as a standby interface (1). However, the IPMP group would still not reflect the original number of two active interfaces because `net1` continues to remain offline (2). Thus, IPMP instead deploys `net2` as an active interface (3).

The `ipmpstat` command displays the post-repair IPMP scenario as follows:

```
# ipmpstat -i
INTERFACE   ACTIVE     GROUP     FLAGS       LINK        PROBE     STATE
net0        yes        itops0    -------     up          ok        ok
net1        no         itops0    --mb-d-     up          ok        offline
net2        yes        itops0    -s-----     up          ok        ok
```

A similar recovery process occurs if the failure involves an active interface that is also configured in `FAILBACK=no` mode, where a failed active interface does not automatically revert to active status upon repair. Suppose that `net0` in Figure 2, “Interface Failure in IPMP,” on page 64 is configured in `FAILBACK=no` mode. In that mode, a repaired `net0` becomes a standby interface, even though it was originally an active interface. The interface `net2` remains active to maintain the IPMP group's original number of two active interfaces.

The `ipmpstat` command displays the recovery information as follows:

```
# ipmpstat -i
INTERFACE    ACTIVE     GROUP     FLAGS      LINK        PROBE     STATE
net0         no         itops0    i------    up          ok        ok
net1         yes        itops0    --mb---    up          ok        ok
net2         yes        itops0    -s-----    up          ok        ok
```

For more information about this type of configuration, see "`FAILBACK=no` Mode" on page 72.

IPMP Addressing

You can configure IPMP failure detection on both IPv4 networks and dual-stack IPv4 and IPv6 networks. Interfaces that you configure with IPMP support two types of addresses: data addresses and test addresses. IP addresses reside on the IPMP interface (the group) *only* and are specified as data addresses. Test addresses are IP addresses that reside on the underlying interfaces.

Data Addresses

Data addresses are the conventional IPv4 and IPv6 addresses that are dynamically assigned to an IP interface at boot time by the DHCP server or manually by using the `ipadm` command. Data addresses are assigned to the IPMP interface (or group) *only*. The standard IPv4 packet traffic and IPv6 packet traffic (if applicable) are considered *data traffic*. Data traffic uses the data addresses that are hosted on the IPMP interface and flow through the active interfaces of that IPMP interface or group.

Test Addresses

Test addresses are IPMP-specific addresses that are used by the `in.mpathd` daemon to perform probe-based failure and repair detection. Test addresses can also be assigned dynamically by the DHCP server or manually by using the `ipadm` command. Only test addresses are assigned to the underlying interfaces of the IPMP group. When an underlying interface fails, the interface's test address continues to be used by the `in.mpathd` daemon for probe-based failure detection to check for the interface's subsequent repair.

Note - Configure test addresses only if you want to use probe-based failure detection. Otherwise, you can enable transitive probing to detect failure without using test addresses. For more information about probe-based failure detection with or without using test addresses, see "Probe-Based Failure Detection" on page 69.

In previous IPMP implementations, test addresses had to be marked as `DEPRECATED` to avoid being used by applications, especially during interface failures. In the current implementation, test addresses reside in the underlying interfaces. Thus, these addresses can no longer be accidentally used by applications that are unaware of IPMP. However, to ensure that these addresses are not considered a possible source for data packets, the system automatically marks any addresses with the `NOFAILOVER` flag as `DEPRECATED`.

You can use any IPv4 address on your subnet as a test address. Because IPv4 addresses are a limited resource for many sites, you might want to use non-routeable RFC 1918 private addresses as test addresses. Note that the `in.mpathd` daemon exchanges only ICMP probes with other hosts on the same subnet as the test address. If you do use RFC 1918-style test addresses, be sure to configure other systems, preferably routers, on the network with addresses on the appropriate RFC 1918 subnet. The `in.mpathd` daemon can then successfully exchange probes with target systems. For more information about RFC 1918 private addresses, refer to RFC 1918, Address Allocation for Private Internets (`http://www.rfc-editor.org/rfc/rfc1918.txt`).

The only valid IPv6 test address is the link-local address of a physical interface. You do not need a separate IPv6 address to serve as an IPMP test address. The IPv6 link-local address is based on the Media Access Control (MAC) address of the interface. Link-local addresses are automatically configured when the interface becomes IPv6-enabled at boot time or when the interface is manually configured through the `ipadm` command.

When an IPMP group has both IPv4 and IPv6 plumbed on all the group's interfaces, you do not need to configure separate IPv4 test addresses. The `in.mpathd` daemon can use the IPv6 link-local addresses as test addresses.

Failure Detection in IPMP

To ensure continuous availability of the network to send or receive traffic, IPMP performs failure detection on the IPMP group's underlying IP interfaces. Failed interfaces remain unusable until they are repaired. Remaining active interfaces continue to function while any existing standby interfaces are deployed as needed.

The `in.mpathd` daemon handles the following types of failure detection:

- Two types of probe-based failure detection:
  - No test addresses are configured (transitive probing).
  - Test addresses are configured.
- Link-based failure detection, if supported by the NIC driver

Probe-Based Failure Detection

Probe-based failure detection consists of using ICMP probes to check whether an interface has failed. The implementation of this failure detection method depends on whether test addresses are used.

Probe-Based Failure Detection Using Test Addresses

This failure detection method involves sending and receiving ICMP probe messages that use test addresses. These messages, also called *probe traffic* or *test traffic*, are sent over the interface to one or more target systems on the same local network. The `in.mpathd` daemon probes all of the targets separately through all the interfaces that have been configured for probe-based failure detection. If no replies are made in response to five consecutive probes on a given interface, the `in.mpathd` daemon considers the interface to have failed. The probing rate depends on the *failure detection time* (*FDT*). The default value for failure detection time is 10 seconds. However, you can tune the FDT in the IPMP configuration file. For instructions, see "How to Configure the Behavior of the IPMP Daemon" on page 92.

To optimize probe-based failure detection, you must set multiple target systems to receive the probes from the `in.mpathd` daemon. By having multiple target systems, you can better determine the nature of a reported failure. For example, the absence of a response from the only defined target system can indicate a failure either in the target system or in one of the IPMP group's interfaces. By contrast, if only one system among several target systems does not respond to a probe, then the failure is likely in the target system rather than in the IPMP group itself.

The `in.mpathd` daemon determines which target systems to probe dynamically. First, the daemon searches the routing table for target systems on the same subnet as the test addresses that are associated with the IPMP group's interfaces. If such targets are found, then the daemon uses them as targets for probing. If no target systems are found on the same subnet, then the daemon sends multicast packets to probe neighbor hosts on the link. The multicast packet is sent to the All Hosts multicast address, `224.0.0.1` in IPv4 and `ff02::1` in IPv6, to determine which hosts to use as target systems. The first five hosts that respond to the echo packets are chosen as targets for probing. If the daemon cannot find routers or hosts that responded to the multicast probes, then the daemon cannot detect probe-based failures. In this case, the `ipmpstat -i` command reports the probe state as `unknown`.

You can use host routes to explicitly configure a list of target systems to be used by the `in.mpathd` daemon. For instructions, see "Configuring Probe-Based Failure Detection" on page 89.

Probe-Based Failure Detection Without Using Test Addresses

With no test addresses, this method is implemented by using two types of probes:

- **ICMP probes**

  ICMP probes are sent by the active interfaces in the IPMP group to probe targets that are defined in the routing table. An *active* interface is an underlying interface that can receive inbound IP packets that are addressed to the interface's link layer (L2) address. The ICMP probe uses the data address as the probe's source address. If the ICMP probe reaches its target and gets a response from the target, then the active interface is operational.
- **Transitive probes**

  Transitive probes are sent by the alternate interfaces in the IPMP group to probe the active interface. An alternate interface is an underlying interface that does not actively receive any inbound IP packets.

  For example, consider an IPMP group that consists of four underlying interfaces and one data address. In this configuration, outbound packets can use all of the underlying interfaces. However, inbound packets can only be received by the interface to which the data address is bound. The remaining three underlying interfaces that cannot receive inbound packets are the *alternate interfaces*.

  If an alternate interface can successfully send a probe to an active interface and receive a response, then the active interface is functional, and by inference, so is the alternate interface that sent the probe.

Note - In Oracle Solaris, probe-based failure detection operates with test addresses. To select probe-based failure detection without test addresses, you must manually enable transitive probing. For instructions, see "Selecting a Failure Detection Method" on page 91.

Group Failure

A *group failure* occurs when all of the interfaces in an IPMP group appear to fail at the same time. In this case, no underlying interface is usable. Also, when all of the target systems fail at the same time and probe-based failure detection is enabled, the `in.mpathd` daemon flushes all of its current target systems and probes for new target systems.

In an IPMP group that has no test addresses, a single interface that can probe the active interface is designated as a *prober*. This designated interface has both the `FAILED` flag and `PROBER` flag set. The data address is bound to this interface, which enables the interface to continue probing the target to detect recovery.

Link-Based Failure Detection

Link-based failure detection is always enabled, provided that the interface supports this type of failure detection.

To determine whether a third-party interface supports link-based failure detection, use the `ipmpstat -i` command. If the output for a given interface includes an `unknown` status in its `LINK` column, then that interface does not support link-based failure detection. Refer to the manufacturer's documentation for more specific information about the network device.

Network drivers that support link-based failure detection monitor the interface's link state and notify the networking subsystem when that link state changes. When notified of a change, the networking subsystem either sets or clears the `RUNNING` flag for that interface, as appropriate. If the `in.mpathd` daemon detects that the interface's `RUNNING` flag has been cleared, the daemon immediately fails the interface.

Failure Detection and the Anonymous Group Feature

IPMP supports failure detection in an anonymous group. By default, IPMP monitors the status only of interfaces that belong to IPMP groups. However, you can configure the IPMP daemon to also track the status of interfaces that do not belong to any IPMP group. Thus, these interfaces are considered to be part of an *anonymous group*. When you issue the `ipmpstat -g` command, the anonymous group is displayed as double-dashes (`--`). In anonymous groups, the interfaces have data addresses that also function as test addresses. Because these interfaces do not belong to a named IPMP group, these addresses are visible to applications. To enable the tracking of interfaces that are not part of an IPMP group, see "How to Configure the Behavior of the IPMP Daemon" on page 92.

Detecting Physical Interface Repairs

Repair detection time is twice the failure detection time. The default time for failure detection is 10 seconds. Accordingly, the default time for repair detection is 20 seconds. After a failed interface has been marked with the `RUNNING` flag again and the failure detection method has detected the interface as repaired, the `in.mpathd` daemon clears the interface's `FAILED` flag. The repaired interface is redeployed, depending on the number of active interfaces that the administrator originally set.

When an underlying interface fails and probe-based failure detection is used, the `in.mpathd` daemon continues probing, either by means of the designated prober when no test addresses are configured or by using the interface's test address.

During an interface repair, how the recovery process proceeds depends on how the failed interface was originally configured, as follows:

- If the failed interface was originally an active interface, the repaired interface reverts to its original active status. The standby interface that functioned as a replacement during the failure is switched back to standby status if enough interfaces are active for the IPMP group, as defined by the system administrator.

  Note - An exception is when the repaired active interface is also configured with the `FAILBACK=no` mode. For more information, see "`FAILBACK=no` Mode" on page 72.

- If the failed interface was originally a standby interface, the repaired interface reverts to its original standby status, provided that the IPMP group reflects the original number of active interfaces. Otherwise, the standby interface becomes an active interface.

For a graphical representation of how IPMP operates during interface failure and repair, see "How IPMP Works" on page 61.

`FAILBACK=no` Mode

By default, active interfaces that have failed and then been repaired automatically become active interfaces in the IPMP group again. This behavior is controlled by the value of the `FAILBACK` parameter in the `in.mpathd` daemon's configuration file. However, even an insignificant disruption that occurs as data addresses are remapped to repaired interfaces might not be acceptable. In this case, you might prefer to enable an activated standby interface to continue as an active interface. IPMP allows you to override the default behavior to prevent an interface from automatically becoming active upon repair. These interfaces must be configured in the `FAILBACK=no` mode. For related procedures, see "How to Configure the Behavior of the IPMP Daemon" on page 92.

When an active interface in `FAILBACK=no` mode fails and is subsequently repaired, the `in.mpathd` daemon restores the IPMP configuration as follows:

- The daemon retains the interface's `INACTIVE` status, provided that the IPMP group reflects the original configuration of active interfaces.
- If the IPMP configuration at the moment of repair does not reflect the group's original configuration of active interfaces, then the repaired interface is redeployed as an active interface, notwithstanding the `FAILBACK=no` status.

Note - The `FAILBACK=NO` mode is set for the whole IPMP group, rather than as a per-interface tunable parameter.

IPMP and Dynamic Reconfiguration

The dynamic reconfiguration (DR) feature of Oracle Solaris enables you to reconfigure system hardware, such as interfaces, while the system is running. DR can be used only on systems that support this feature. On systems that support DR, IPMP is integrated into the Reconfiguration Coordination Manager (RCM) framework. Thus, you can safely attach, detach, or reattach NICs and RCM manages the dynamic reconfiguration of system components. For example, you can attach, plumb, and then add new interfaces to existing IPMP groups. After these interfaces are configured, they are immediately available for use by IPMP.

All requests to detach NICs are first checked to ensure that connectivity can be preserved. For example, by default you cannot detach a NIC that is not in an IPMP group. You also cannot detach a NIC that contains the only functioning interfaces in an IPMP group. However, if you must remove the system component, you can override this behavior by using the `-f` option of the `cfgadm` command, as described in the `cfgadm`(1M) man page.

If the checks are successful, the `in.mpathd` daemon sets the `OFFLINE` flag for the interface. All test addresses on the interfaces are unconfigured. Then, the NIC is unplumbed from the system.

If any of these steps fail, or if the DR of other hardware on the same system component fails, only the persistent configuration is restored. In this case, the following error message is logged:

```
"IP: persistent configuration is restored for <ifname>"
```

Otherwise, the detach request completes successfully. You can remove the component from the system and no existing connections are disrupted.

Note - When replacing NICs, make sure that the replacement cards are of the same type, such as Ethernet. After the NIC is replaced, then the persistent IP interface configurations are applied to that NIC.

♦ ♦ ♦ CHAPTER 3

Administering IPMP

This chapter describes how to administer interface groups with IPMP in the Oracle Solaris release. The tasks that are in this chapter describe how to configure IPMP by using the `ipadm` command, which replaces the `ifconfig` command that is used to configure IPMP in Oracle Solaris 10. To find out more about how these two commands map to each other, see "Comparing the ifconfig Command to the ipadm Command" in *Transitioning From Oracle Solaris 10 to Oracle Solaris 11.3*. See also the `ifconfig`(5) man page.

For a detailed explanation of changes in the IPMP conceptual model, see "What's New in IPMP" on page 55.

This chapter contains the following topics:

Configuring IPMP Groups

The following information describes how to plan and configure IPMP groups. The overview in Chapter 2, "About IPMP Administration" describes the implementation of an IPMP group as an interface. In this chapter, the terms *IPMP group* and *IPMP interface* are used interchangeably.

This section contains the following tasks:

▼ How to Plan an IPMP Group

The following procedure includes the required planning tasks and information to be gathered prior to configuring an IPMP group. You do not need to perform these tasks in sequential order.

Your IPMP configuration depends on the network requirements for handling the type of traffic that is hosted on your system. IPMP spreads outbound network packets across the IPMP group's interfaces and thus improves network throughput. For inbound traffic, each connection must travel through the same underlying interface as well, to avoid out-of-order packets.

Thus, if your network handles a huge volume of outbound traffic, configuring several interfaces into an IPMP group only improves network performance if multiple connections also exist. For inbound traffic, if that traffic is destined for the different IP addresses that are hosted by the IPMP group interface, having more than one underlying interface can help performance because inbound load spreading is based on IP address.

Note - You must configure only one IPMP group for each subnet or L2 broadcast domain. For more information, see "Rules for Using IPMP" on page 58.

---

1. **Determine the general IPMP configuration that suits your needs.**

   Refer to the information in the task summary of this procedure for guidance in determining which IPMP configuration to use.

2. **(SPARC only) Verify that each interface in the group has a unique MAC address.**

   To configure a unique MAC address for each interface on the system, see "How to Ensure That the MAC Address of Each Interface Is Unique" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.

3. **Ensure that the same set of STREAMS modules is configured and pushed on all interfaces in the IPMP group.**

   For guidelines and the command syntax to use, see "Rules for Using IPMP" on page 58.

4. **Use the same IP addressing format on all of the interfaces in the IPMP group.**

   If one interface is configured for IPv4, then you must configure all of the interfaces in the IPMP group for IPv4. Likewise, if you add IPv6 addressing to one interface, then you must configure all of the interfaces in the IPMP group for IPv6 support.

5. **Determine the type of failure detection that you want to implement.**

   For example, if you want to implement probe-based failure detection, then you must configure test addresses on the underlying interfaces. See "Failure Detection in IPMP" on page 68.

6. **Ensure that all of the interfaces in the IPMP group are connected to the same local network.**

For example, you can configure Ethernet switches on the same IP subnet into an IPMP group. You can configure any number of interfaces into an IPMP group.

Note - You can also configure a single-interface IPMP group, for example, if your system has only one physical interface. See "Types of IPMP Interface Configurations" on page 60.

---

7. **Ensure that the IPMP group does not contain interfaces with different network media types.**

   The interfaces that are grouped together must be of the same interface type. For example, you cannot combine Ethernet and Token Ring interfaces in an IPMP group. As another example, you cannot combine a Token bus interface with asynchronous transfer mode (ATM) interfaces in the same IPMP group.

8. **For IPMP with ATM interfaces, configure the ATM interfaces in LAN emulation mode.**

   IPMP is not supported for interfaces using Classical IP over ATM technology as defined in RFC 1577 (`http://www.rfc-editor.org/rfc/rfc1577.txt`) and RFC 2225 (`http://www.rfc-editor.org/rfc/rfc2225.txt`).

▼ How to Configure an IPMP Group That Uses DHCP

You can configure a multiple-interfaced IPMP group with active-active interfaces or active-standby interfaces. See "Types of IPMP Interface Configurations" on page 60. The following procedure describes how to configure an active-standby IPMP group with DHCP.

Before You Begin Before performing the following procedure, do the following:

- Ensure that the IP interfaces that will be in the prospective IPMP group have been correctly configured over the system's network datalinks. For procedures, see *Configuring and Managing Network Components in Oracle Solaris 11.3*. You can create an IPMP interface even if you have not created the underlying IP interfaces. However, without creating underlying IP interfaces, subsequent configurations on the IPMP interface will fail.
- Additionally, if you are using a SPARC based system, you must configure a unique MAC address for each interface. See "How to Ensure That the MAC Address of Each Interface Is Unique" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.
- Finally, if you are using DHCP, make sure that the underlying interfaces have *infinite leases*. Otherwise, if an IPMP group failure occurs, the test addresses will expire and the `in.mpathd` daemon will then disable probe-based failure detection and link-based failure detection will then be used. If link-based failure detection discovers that the interface is functioning, the daemon might erroneously report that the interface has been repaired. For more information about configuring DHCP, see *Working With DHCP in Oracle Solaris 11.3*.

1. **Become the `root` role.**

2. **Create an IPMP interface.**

   ```
   # ipadm create-ipmp ipmp-interface
   ```

   where *ipmp-interface* specifies the name of the IPMP interface. You can assign any meaningful name to the IPMP interface. As with any IP interface, the name consists of a string and a number, for example, `ipmp0`.

3. **Create the underlying IP interfaces, if they do not yet exist.**

   ```
   # ipadm create-ip under-interface
   ```

   where *under-interface* refers to the IP interface that you will add to the IPMP group.

4. **Add the underlying IP interfaces that will contain test addresses for the IPMP group.**

   ```
   # ipadm add-ipmp -i under-interface1 [-i under-interface2 ...] ipmp-interface
   ```

   You can add as many IP interfaces to the IPMP group as are available on the system.

5. **Specify that DHCP configure and manage the data addresses on the IPMP interface.**

   ```
   # ipadm create-addr -T dhcp ipmp-interface
   ```

   The previous step associates the address that is provided by the DHCP server with an address object. The address object uniquely identifies the IP address by using the format *interface*/*address-type*, for example, `ipmp0/v4`. For more information about the address object, see "How to Configure an IPv4 Interface" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.

6. **If you use probe-based failure detection with test addresses, specify that DHCP manage the test addresses on the underlying interfaces.**

   ```
   # ipadm create-addr -T dhcp under-interface
   ```

   The address object that is automatically created in Step 6 uses the format *under-interface*/*address-type*, for example, `net0/v4`.

7. **(Optional) Repeat Step 6 for each underlying interface of the IPMP group.**

Example 17 Configuring an IPMP Group With DHCP

The following example shows the configuration of an active-standby IPMP group with DHCP and is based on the following scenario:

- Three underlying interfaces `net0`, `net1`, and `net2` are configured into an IPMP group.
- The IPMP interface `ipmp0` shares the same name with the IPMP group.
- `net2` is the designated standby interface.

- All of the underlying interfaces are assigned test addresses.

The IPMP interface is first created.

```
# ipadm create-ipmp ipmp0
```

The underlying IP interfaces are created and added to the IPMP interface.

```
# ipadm create-ip net0
# ipadm create-ip net1
# ipadm create-ip net2

# ipadm add-ipmp -i net0 -i net1 -i net2 ipmp0
```

The DHCP-managed IP addresses are assigned to the IPMP interface. IP addresses that are assigned to the IPMP interface are data addresses. In this example, the IPMP interface has two data addresses.

```
# ipadm create-addr -T dhcp ipmp0
ipadm: ipmp0/v4
# ipadm create-addr -T dhcp ipmp0
ipadm: ipmp0/v4a
```

Then, the DHCP-managed IP addresses are assigned to the underlying IP interfaces of the IPMP group. IP addresses that are assigned to the underlying interfaces are test addresses that are to be used for probe-based failure detection.

```
# ipadm create-addr -T dhcp net0
ipadm: net0/v4
# ipadm create-addr -T dhcp net1
ipadm: net1/v4
# ipadm create-addr -T dhcp net2
ipadm net2/v4
```

Lastly, the `net2` interface is configured as a standby interface.

```
# ipadm set-ifprop -p standby=on -m ip net2
```

▼ How to Configure an Active-Active IPMP Group

The following procedure describes how to manually configure an active-active IPMP group. In this procedure, Steps 1-4 describe how to configure a link-based active-active IPMP group. Step 5 describes how to make the link-based configuration probe-based.

Before You Begin Ensure that the IP interfaces that will be in the prospective IPMP group are correctly configured over the system's network datalinks. For instructions see "How to Configure an IPv4 Interface" in *Configuring and Managing Network Components in Oracle Solaris 11.3*. You can create an IPMP interface even if the underlying IP interfaces do not yet exist. However, subsequent configurations on the IPMP interface will fail.

Additionally, if you are using a SPARC based system, configure a unique MAC address for each interface. See "How to Ensure That the MAC Address of Each Interface Is Unique" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.

1. **Become the `root` role.**
2. **Create an IPMP interface.**

   ```
   # ipadm create-ipmp ipmp-interface
   ```

   where *ipmp-interface* specifies the name of the IPMP interface. You can assign any meaningful name to the IPMP interface. As with any IP interface, the name consists of a string and a number, for example, `ipmp0`.
3. **Add the underlying IP interfaces to the group.**

   ```
   # ipadm add-ipmp -i under-interface1 [-i underinterface2 ...] ipmp-interface
   ```

   where *under-interface* refers to the underlying interface of the IPMP group. You can add as many IP interfaces as are available on the system.

   Note - In a dual-stack environment, placing the IPv4 instance of an interface under a particular group automatically places the IPv6 instance under the same group.

   ---
4. **Add the data addresses to the IPMP interface.**

   ```
   # ipadm create-addr -a address ipmp-interface
   ```

   where *address* can be in CIDR notation.

   Note - Only the DNS address of the IPMP group name or IP address is required.

   ---
5. **If you use probe-based failure detection with test addresses, add the test addresses on the underlying interfaces.**

   ```
   # ipadm create-addr -a address under-interface
   ```

   where *address* can be in CIDR notation. All test IP addresses in an IPMP group must belong to a single IP subnet and therefore using same network prefix.

▼ How to Configure an Active-Standby IPMP Group

The following procedure describes how to configure an IPMP group in which one interface is kept as a standby interface. This interface is deployed only when an active interface in the group fails.

For overview information about standby interfaces, see "Types of IPMP Interface Configurations" on page 60.

1. **Become the `root` role.**

2. **Create an IPMP interface.**

   ```
   # ipadm create-ipmp ipmp-interface
   ```

   where *ipmp-interface* specifies the name of the IPMP interface.

3. **Add the underlying IP interfaces to the group.**

   ```
   # ipadm add-ipmp -i under-interface1 [-i underinterface2 ...] ipmp-interface
   ```

   where *under-interface* refers to the underlying interface of the IPMP group. You can add as many IP interfaces as are available on the system.

   Note - In a dual-stack environment, placing the IPv4 instance of an interface under a particular IPMP group automatically places the IPv6 instance under the same group.

   ---

4. **Add the data addresses to the IPMP interface.**

   ```
   # ipadm create-addr -a address ipmp-interface
   ```

   where *address* can be in CIDR notation.

5. **If you use probe-based failure detection with test addresses, add the test addresses on the underlying interfaces.**

   ```
   # ipadm create-addr -a address under-interface
   ```

   where *address* can be in CIDR notation. All test IP addresses in an IPMP group must belong to a single IP subnet and therefore using same network prefix.

6. **Configure one of the underlying interfaces as a standby interface.**

   ```
   # ipadm set-ifprop -p standby=on -m ip under-interface
   ```

Example 18 Configuring an Active-Standby IPMP Group

The following example shows how to create an active-standby IPMP configuration.

First, the IPMP interface is created.

```
# ipadm create-ipmp ipmp0
```

The underlying IP interfaces are then created and added to the IPMP interface.

```
# ipadm create-ip net0
# ipadm create-ip net1
# ipadm create-ip net2

# ipadm add-ipmp -i net0 -i net1 -i net2 ipmp0
```

Next, IP addresses are assigned to the IPMP interface. IP addresses that are assigned to the IPMP interface are data addresses. In this example, the IPMP interface has two data addresses.

```
# ipadm create-addr -a 192.168.10.10/24 ipmp0
ipadm: ipmp0/v4
# ipadm create-addr -a 192.168.10.15/24 ipmp0
ipadm: ipmp0/v4a
```

The IP address in this example includes the `prefixlen` property, which is expressed as a decimal number. The `prefixlen` portion of the IP address specifies the number of left-most contiguous bits of the address that comprise the IPv4 netmask or the IPv6 prefix of the address. The remaining low-order bits define the host part of the address. When the `prefixlen` property is converted to a text representation of the address, the address contains 1's for the bit positions that are to be used for the network part and 0's for the host part. This property is not supported on the `dhcp` address object type. For more information, see the `ipadm`(1M) man page.

IP addresses are then assigned to the underlying IP interfaces of the IPMP group. IP addresses that are assigned to the underlying interfaces are test addresses to be used for probe-based failure detection.

```
# ipadm create-addr -a 192.168.10.30/24 net0
ipadm: net0/v4
# ipadm create-addr -a 192.168.10.32/24 net1
ipadm: net1/v4
# ipadm create-addr -a 192.168.10.34/24 net2
ipadm: net2/v4
```

Lastly, the `net2` interface is configured as a standby interface.

```
# ipadm set-ifprop -p standby=on -m ip net2
```

The administrator can view the IPMP configuration by using the `ipmpstat` command.

```
# ipmpstat -g
GROUP     GROUPNAME   STATE      FDT        INTERFACES
ipmp0     ipmp0      ok         10.00s     net0 net1 (net2)

# ipmpstat -t
INTERFACE  MODE     TESTADDR        TARGETS
net0       routes   192.168.10.30   192.168.10.1
net1       routes   192.168.10.32   192.168.10.1
net2       routes   192.168.10.34   192.168.10.5
```

Maintaining IP Connectivity and Routing While Deploying IPMP

You can add an IP interface to an IPMP group by using either the `ipadm` command or the `ifconfig` command. Due to backward compatibility with previous versions of Oracle Solaris IPMP, when you use the `ifconfig` command, any data addresses that are not marked with `IFF_NOFAILOVER` are migrated to the IPMP interface that is associated with the IPMP group. However, when you add an IP interface to an IPMP group by using the `ipadm` command, any address that is currently configured on the IP interface becomes a test address for that IP interface, meaning the address is not migrated to the IPMP interface as a data address.

If you want the IP address to be an IPMP data address, you must first remove the address from the IP interface and then reconfigure the address directly on the IPMP interface, as shown in the following example:

```
# ipadm
NAME    CLASS/TYPE  STATE    UNDER    ADDR
...
ipmp0   ipmp        down      --      --
net0    ip          ok        ipmp0   --
net0/v4 static      ok        –       192.168.11.1/24

# ipadm delete-addr net0/v4
# ipadm create-addr -T static -a local=192.168.11.1/24 ipmp0/v4

# ipadm
NAME      CLASS/TYPE  STATE    UNDER     ADDR
...
ipmp0     ipmp        ok       --        --
ipmp0/v4  static      ok       --        192.168.11.1/24
net0      ip          ok       ipmp0     --
```

Also, be mindful that any routes that you defined by using specific IP interfaces will no longer work if these interfaces are subsequently added to an IPMP group. To ensure that a default route is preserved while using IPMP, you can define the route without specifying an interface. Using this method ensures that any interface, including an IPMP interface, can be used for routing, thereby enabling the system to continue to route traffic.

Loss of routing when configuring IPMP can also occur in association with an Oracle Solaris installation. During the installation, you are required to define a default route, for which you can use an interface on the system, such as the primary interface. Subsequently, if you configure an IPMP group by using the same interface on which you defined the default route, the system can no longer route network packets because the interface's address has been transferred to the IPMP interface. The following procedure describes a method for preserving the default route when using IPMP.

▼ How to Preserve the Default Route While Using IPMP

The following task assumes the primary interface on the system is the interface on which the default route is defined. This type of routing loss applies to any interface that is used for routing, which later becomes part of an IPMP group.

1. **Log in to the system by using a console.**
   You must use the console to perform this procedure. If you use the `ssh` or `telnet` command to log in, the connection is lost when you perform the subsequent steps.

2. **(Optional) Display the routes that are currently defined in the routing table.**

   ```
   # netstat -nr
   ```

3. **Delete the route that is bound to the specific interface.**

   ```
   # route -p delete default gateway-address -ifp interface
   ```

4. **Add the route without specifying an interface.**

   ```
   # route -p add default gateway-address
   ```

5. **(Optional) Display the redefined routes in the routing table.**

   ```
   # netstat -nr
   ```

6. **(Optional) If the information has not changed, restart the routing service, then recheck the information in the routing table to make sure the routes have been correctly redefined.**

   ```
   # svcadm restart routing-setup
   ```

Example 19 Defining Routes for IPMP

This example assumes that the default route was defined for `net0` during the installation.

```
# netstat -nr
Routing Table: IPv4
Destination     Gateway       Flags    Ref     Use       Interface
-------------  ------------  --------  -----  -----------  --------
default        10.153.125.1     UG      107    176682262   net0
10.153.125.0   10.153.125.222   U        22    137738792   net0

# route -p delete default 10.153.125.1 -ifp net0
# route -p add default 10.153.125.1

# netstat -nr
Routing Table: IPv4
```

```
Destination     Gateway       Flags    Ref     Use        Interface
-------------  ------------  --------  -----  -----------  --------
default        10.153.125.1     UG      107    176682262
10.153.125.0   10.153.125.222   U        22    137738792    net0
```

Administering IPMP

This section contains the following procedures for maintaining an IPMP group that you have created on the system:

- "How to Add an Interface to an IPMP Group" on page 85
- "How to Remove an Interface From an IPMP Group" on page 86
- "How to Add IP Addresses to an IPMP Group" on page 86
- "How to Delete IP Addresses From an IPMP Interface" on page 87
- "How to Move an Interface From One IPMP Group to Another IPMP Group" on page 88
- "How to Delete an IPMP Group" on page 89

▼ How to Add an Interface to an IPMP Group

Before You Begin Ensure that the interface that you add to the group meets all of the necessary requirements. For a list of requirements, see "How to Plan an IPMP Group" on page 76.

1. **Become the `root` role.**

2. **If the underlying IP interface does not yet exist, create the interface.**

   ```
   # ipadm create-ip under-interface
   ```

3. **Add the IP interface to the IPMP group.**

   ```
   # ipadm add-ipmp -i under-interface ipmp-interface
   ```

   where *ipmp-interface* refers to the IPMP group to which you want to add the underlying interface.

Example 20 Adding an Interface to an IPMP Group

The following example shows how you would add the `net4` interface to the IPMP group `ipmp0`.

```
# ipadm create-ip net4
# ipadm add-ipmp -i net4 ipmp0
# ipmpstat -g
GROUP    GROUPNAME    STATE      FDT       INTERFACES
```

```
ipmp0   ipmp0       ok         10.00s    net0 net1 net4
```

▼ How to Remove an Interface From an IPMP Group

1. **Become the `root` role.**

2. **Remove one or more interfaces from the IPMP group.**

   `# ipadm remove-ipmp -i` *under-interface*`[ -i` *under-interface* `...]` *ipmp-interface*

   where *under-interface* refers to an IP interface that you are removing from the IPMP group and *ipmp-interface* refers to the IPMP group from which you are removing underlying interfaces.

   You can remove as many underlying interfaces in a single command, as required. Removing all of the underlying interfaces does not delete the IPMP interface. Instead, it exists as an empty IPMP interface or group.

Example 21 Removing an Interface From an IPMP Group

The following example shows how to remove the `net4` interface from the IPMP group `ipmp0`.

```
# ipadm remove-ipmp -i net4 ipmp0
# ipmpstat -g
GROUP   GROUPNAME   STATE      FDT       INTERFACES
ipmp0   ipmp0      ok         10.00s    net0 net1
```

▼ How to Add IP Addresses to an IPMP Group

To add IP addresses to an IPMP group, use the `ipadm create-addr` command. For IPMP configuration, an IP address can be either a data address or a test address. A data address is added to an IPMP interface, while a test address is added to an underlying interface of the IPMP interface. The following procedure describes how to add IP addresses that are either test addresses or data addresses.

1. **Become the `root` role.**

2. **Add the IP addresses to an IPMP group.**

   - **Add data addresses to an IPMP group as follows:**

     `# ipadm create-addr -a` *address ipmp-interface*

     An address object is automatically assigned to the IP address that you just created. An address object is a unique identifier of the IP address. The address object's name uses the naming convention *interface*/*random-string*. Thus, address objects of data addresses would include the IPMP interface in their names.

- **Add test addresses to an underlying interface of an IPMP group as follows:**

  ```
  # ipadm create-addr -a address under-interface
  ```

  An address object is automatically assigned to the IP address that you just created. An address object is a unique identifier of the IP address. The address object's name uses the naming convention *interface*/*random-string*. Thus, address objects of test addresses would include the underlying interface in their names.

How to Delete IP Addresses From an IPMP Interface

To delete IP addresses from an IPMP group, use the `ipadm delete-addr` command. For IPMP configuration, data addresses are hosted on the IPMP interface and test addresses are hosted on underlying interfaces. The following procedure shows how to remove IP addresses that are either data addresses or test addresses.

1. **Become the `root` role.**
2. **Determine the IP addresses that you want to remove.**
   - **Display a list of data addresses as follows:**

     ```
     # ipadm show-addr ipmp-interface
     ```

   - **Display a list of test addresses as follows:**

     ```
     # ipadm show-addr
     ```

     Test addresses are identified by address objects whose names include the underlying interfaces where the addresses are configured.
3. **Remove the IP addresses from an IPMP group.**
   - **Remove data addresses as follows:**

     ```
     # ipadm delete-addr addrobj
     ```

     where *addrobj* must include the name of the IPMP interface. If the address object that you type does not include the IPMP interface name, then the address that will be deleted is not a data address.

- **Remove test addresses as follows:**

  ```
  # ipadm delete-addr addrobj
  ```

  where *addrobj* must include the name of the correct underlying interface to delete the correct test address.

Example 22 Removing a Test Address From an Interface

The following example uses the configuration of the active-standby IPMP group `ipmp0` that is shown in Example 18, "Configuring an Active-Standby IPMP Group," on page 81. This example removes the test address from the underlying interface `net1`.

```
# ipadm show-addr net1
ADDROBJ           TYPE     STATE     ADDR
net1/v4           static   ok        192.168.10.30

# ipadm delete-addr net1/v4
```

▼ How to Move an Interface From One IPMP Group to Another IPMP Group

You can place an interface in a new IPMP group when the interface belongs to an existing IPMP group. You do not need to remove the interface from the current IPMP group. When you place the interface in a new group, the interface is automatically removed from any existing IPMP group.

1. **Become the `root` role.**

2. **Move the interface to a new IPMP group.**

   ```
   # ipadm add-ipmp -i under-interface ipmp-interface
   ```

   where *under-interface* refers to the underlying interface that you want to move and *ipmp-interface* refers to the IPMP interface to which you want to move the underlying interface.

Example 23 Moving an Interface to a Different IPMP Group

In the following example, the underlying interfaces of the IPMP group are `net0`, `net1`, and `net2`. The example shows how to remove the `net0` interface from IPMP group `ipmp0` and then places `net0` in the IPMP group `cs-link1`.

```
# ipadm add-ipmp -i net0 ca-link1
```

▼ How to Delete an IPMP Group

Use the following procedure if you no longer need a specific IPMP group.

1. **Become the `root` role.**

2. **Identify the IPMP group and the underlying IP interfaces that are to be deleted.**

   ```
   # ipmpstat -g
   ```

3. **Remove all of the IP interfaces that currently belong to the IPMP group.**

   ```
   # ipadm remove-ipmp -i under-interface[, -i under-interface, ...] ipmp-interface
   ```

   where *under-interface* refers to the underlying interface that you want to remove and *ipmp-interface* refers to the IPMP interface from which you want to remove the underlying interface.

   Note - To successfully delete an IPMP interface, no IP interface must exist as part of the IPMP group.

   ---

4. **Delete the IPMP interface.**

   ```
   # ipadm delete-ipmp ipmp-interface
   ```

   After you delete the IPMP interface, any IP address that is associated with the interface is also deleted from the system.

Example 24 Deleting an IPMP Interface

The following example deletes the interface `ipmp0` with the underlying IP interface `net0` and `net1`.

```
# ipmpstat -g
GROUP   GROUPNAME   STATE      FDT        INTERFACES
ipmp0   ipmp0      ok         10.00s     net0 net1

# ipadm remove-ipmp -i net0 -i net1 ipmp0

# ipadm delete-ipmp ipmp0
```

Configuring Probe-Based Failure Detection

This section contains the following topics:

- "About Probe-Based Failure Detection" on page 90

- "Requirements for Choosing Targets for Probe-based Failure Detection" on page 91
- "Selecting a Failure Detection Method" on page 91
- "How to Manually Specify Target Systems for Probe-Based Failure Detection" on page 92
- "How to Configure the Behavior of the IPMP Daemon" on page 92

About Probe-Based Failure Detection

Probe-based failure detection involves the use of target systems, as described in "Probe-Based Failure Detection" on page 69. In identifying targets for probe-based failure detection, the `in.mpathd` daemon operates in two modes: *router target mode* or *multicast target mode*. In router target mode, the daemon probes targets that are defined in the routing table. If no targets are defined, then the daemon operates in multicast target mode, where multicast packets are sent out to probe neighbor hosts on the LAN.

Preferably, you should set up target systems for the `in.mpathd` daemon to probe. For some IPMP groups, the default router is sufficient as a target. However, for some IPMP groups, you might want to configure specific targets for probe-based failure detection. To specify the targets, set up host routes in the routing table as probe targets. Any host routes that are configured in the routing table are listed before the default router. IPMP uses the explicitly defined host routes for target selection. Thus, you should set up host routes to configure specific probe targets rather than use the default router.

To set up host routes in the routing table, you use the `route` command. You can use the `-p` option with this command to add persistent routes. For example, `route -p add` adds a route that will remain in the routing table even after you reboot the system. The `-p` option thus enables you to add persistent routes without needing any special scripts to recreate these routes with every system start-up. To optimally use probe-based failure detection, make sure that you set up multiple targets to receive probes.

The `route` command operates on both IPv4 and IPv6 routes, with IPv4 routes as the default. If you use the `-inet6` option immediately after the `route` command, operations are performed on IPv6 routes.

The procedure "How to Manually Specify Target Systems for Probe-Based Failure Detection" on page 92 shows the exact syntax to use to add persistent routes to targets for probe-based failure detection. For more information about the options that can be used with the `route` command, see the `route`(1M) man page and "Maintaining IP Connectivity and Routing While Deploying IPMP" on page 83.

Requirements for Choosing Targets for Probe-based Failure Detection

Refer to the following requirements to determine which hosts on your network might serve as good targets:

- Make sure that the prospective targets are available and running. Make a list of their IP addresses.
- Make sure that the target interfaces are on the same network as the IPMP group that you are configuring.
- The netmask and broadcast addresses of the target systems must be the same as the addresses in the IPMP group.
- The target system must be able to answer ICMP requests from the interface that is using probe-based failure detection.

Selecting a Failure Detection Method

Probe-based failure detection can operate either by using a transitive method that does not use test addresses or by configuring test addresses.

Also, if the NIC driver supports it, link-based failure detection is always enabled automatically. You cannot disable link-based failure detection if this method is supported by the NIC driver. However, you can select which type of probe-based failure detection to implement.

Before selecting a probe-based detection method, take sure that your probe targets meet the requirements that are listed in "Requirements for Choosing Targets for Probe-based Failure Detection" on page 91.

To use just transitive probing, do the following:

1. Enable the IPMP property `transitive-probing` by using SMF commands.

   ```
   # svccfg -s svc:/network/ipmp setprop config/transitive-probing=true
   # svcadm refresh  svc:/network/ipmp:default
   ```

   For more information about setting this property, see the `in.mpathd`(1M) man page.
2. Remove any existing test addresses that have been configured for the IPMP group.

   ```
   # ipadm delete-addr address addrobj
   ```

   where *addrobj* must refer to an underlying interface that hosts a test address.

To use test addresses to probe for failure, do the following:

Assign test addresses to the underlying interfaces of the IPMP group.

```
# ipadm create-addr -a address under-interface
```

where *address* can be in CIDR notation and *under-interface* is an underlying interface of the IPMP group.

▼ How to Manually Specify Target Systems for Probe-Based Failure Detection

The following procedure describes how to add specific targets if you are using test addresses to implement probe-based failure detection.

Before You Begin Make sure that your probe targets meet the requirements that are listed in "Requirements for Choosing Targets for Probe-based Failure Detection" on page 91.

1. **Log in with your user account to the system on which you are configuring probe-based failure detection.**

2. **Add a route to the particular host that is to be used as a target in probe-based failure detection.**

   ```
   % route -p add -host destination-IP gateway-IP -static
   ```

   where *destination-IP* and *gateway-IP* are IPv4 addresses of the host to be used as a target. For example, you would type the following to specify the target system `192.168.10.137`, which is on the same subnet as the interfaces in IPMP group `ipmp0`:

   ```
   % route -p add -host 192.168.10.137 192.168.10.137 -static
   ```

   This new route will be automatically configured every time the system is restarted. If you only want to define a temporary route to a target system for probe-based failure detection, then do not use the `-p` option.

3. **Add routes to additional hosts on the network that are to be used as target systems.**

▼ How to Configure the Behavior of the IPMP Daemon

Use the IPMP `/etc/default/mpathd` configuration file to configure the following system-wide parameters for IPMP groups:

- `FAILURE_DETECTION_TIME`

- `FAILBACK`
- `TRACK_INTERFACES_ONLY_WITH_GROUPS`

1. **Become the `root` role.**

2. **Edit the `/etc/default/mpathd` file.**

   ```
   # pfedit /etc/default/mpathd
   ```

   See the `pfedit`(1M) man page for instructions.

   Change the default value of one or more of the following three parameters:

   - Type the new value for the `FAILURE_DETECTION_TIME` parameter as follows:

     ```
     FAILURE_DETECTION_TIME=n
     ```

     where *n* is the amount of time in seconds for ICMP probes to detect whether an interface failure has occurred. The default is 10 seconds.

   - Type the new value for the `FAILBACK` parameter as follows:

     ```
     FAILBACK=[yes | no]
     ```

`yes`	Is the default for the failback behavior of IPMP. When the repair of a failed interface is detected, network access fails back to the repaired interface, as described in "Detecting Physical Interface Repairs" on page 71.
`no`	Indicates that data traffic does not return to a repaired interface. When a failed interfaces is detected as repaired, the `INACTIVE` flag is set for that interface. This flag indicates that the interface is currently not to be used for data traffic. The interface can still be used for probe traffic. For example, assume that the IPMP group `ipmp0` consists of two interfaces, `net0` and `net1`. In the `/etc/default/mpathd` file, the `FAILBACK=no` parameter is set. If `net0` fails, then it is flagged as `FAILED` and becomes unusable. After repair, the interface is flagged as `INACTIVE` and remains unusable because of the `FAILBACK=no` value. If `net1` fails and only `net0` is in the `INACTIVE` state, then the `INACTIVE` flag for `net0` is cleared and the interface becomes usable. If the IPMP group has other interfaces that are also in the `INACTIVE` state, then any one of these `INACTIVE` interfaces, and not necessarily `net0`, can be cleared and become usable when `net1` fails.

   - Type the new value for the `TRACK_INTERFACES_ONLY_WITH_GROUPS` parameter as follows:

     ```
     TRACK_INTERFACES_ONLY_WITH_GROUPS=[yes | no]
     ```

`yes`	Is the default for the behavior of IPMP. This value causes IPMP to ignore network interfaces that are not configured into an IPMP group.
`no`	Sets failure and repair detection for *all* network interfaces, regardless of whether they are configured into an IPMP group. However, when a failure or repair is detected on an interface that is not configured into an IPMP group, no action is triggered in IPMP to maintain the networking functions of that interface. Therefore, the `no` value is only useful for reporting failures and does not directly improve network availability. For more information about this parameter and the anonymous group feature, see "Failure Detection and the Anonymous Group Feature" on page 71.

3. **Restart the `in.mpathd` daemon.**

```
# pkill -HUP in.mpathd
```

The daemon restarts with the new parameter values in effect.

Monitoring IPMP Information

The following examples show how to use the `ipmpstat` command to monitor different aspects of the IPMP groups that are on the system. You can observe the status of an IPMP group as a whole or its underlying IP interfaces. You can also verify the configuration of data and test addresses for an IPMP group. You can also use the same command to obtain information about failure detection. For more information, see the `ipmpstat(1M)` man page.

When you use the `ipmpstat` command, by default, the most meaningful fields that fit in 80 columns are displayed. In the output, all of the fields that are specific to the option that you use with the `ipmpstat` command are displayed, except in the case where the `ipmpstat` is used with the `-p` option.

By default, host names are displayed in the output instead of numeric IP addresses, provided that the host names exist. To list the numeric IP addresses in the output, use the `-n` option with other options to display specific IPMP group information.

Note - In the following examples, use of the `ipmpstat` command does not require system administrator privileges, unless stated otherwise.

Use the `ipmpstat` command with the following options to display the desired information:

`-g`	Displays information about the IPMP groups on the system. See Example 25, "Obtaining IPMP Group Information," on page 95.
`-a`	Displays the data addresses that are configured for the IPMP groups. See Example 26, "Obtaining IPMP Data Address Information," on page 96.
`-i`	Displays information about IP interfaces that are related to IPMP configuration. See Example 27, "Obtaining Information About Underlying IP Interfaces of an IPMP Group," on page 97.
`-t`	Displays information about target systems that are used for detecting failure. This option also displays the test addresses that are used by the IPMP group. See Example 28, "Obtaining IPMP Probe Target Information," on page 99.
`-p`	Displays information about the probes that are being used for failure detection. See Example 29, "Observing IPMP Probes," on page 100.

The following additional examples show how to display information about your system's IPMP configuration by using the `ipmpstat` command.

EXAMPLE 25 Obtaining IPMP Group Information

The `-g` option displays the status of the various IPMP groups that are on the system, including the status of their underlying interfaces. If probe-based failure detection is enabled for a specific group, the command also includes the failure detection time for that group.

```
% ipmpstat -g
GROUP    GROUPNAME   STATE      FDT       INTERFACES
ipmp0    ipmp0       ok         10.00s    net0 net1
acctg1   acctg1      failed     --        [net3 net4]
field2   field2      degraded   20.00s    net2 net5 (net7) [net6]
```

The output fields provide the following information:

`GROUP` Specifies the IPMP interface name. For an anonymous group, this field is empty. For more information about anonymous groups, see the `in.mpathd`(1M) man page.

`GROUPNAME` Specifies the name of the IPMP group. In the case of an anonymous group, this field is empty.

`STATE` Indicates an IPMP group's current status, which can be one of the following:

- `ok` – Indicates that all of the underlying interfaces of the IPMP group are usable.

- `degraded` – Indicates that some of the underlying interfaces in the group are unusable.
- `failed` – Indicates that all of the group's interfaces are unusable.

`FDT`	Specifies the failure detection time, if failure detection is enabled. If failure detection is disabled, this field is empty.

`INTERFACES`

Specifies the underlying interfaces that belong to the IPMP group. In this field, active interfaces are displayed first, then inactive interfaces, and finally unusable interfaces. The status of an interface is indicated by the manner in which it is displayed:

- *interface* (without parentheses or square brackets) – Indicates an active interface. Active interfaces are being used by the system to send or receive data traffic.
- *(interface)* (with parentheses) – Indicates a functioning but inactive interface. The interface is not in use, as defined by administrative policy.
- *[interface]* (with square brackets) – Indicates that the interface is unusable because it has either failed or been taken offline.

EXAMPLE 26 Obtaining IPMP Data Address Information

The `-a` option displays data addresses and the IPMP group to which each address belongs. The displayed information also includes those addresses that are available for use, depending on whether the addresses have been toggled by the `ipadm [up-addr/down-addr]` command. You can also determine on which inbound or outbound interface an address can be used.

```
% ipmpstat -an
ADDRESS         STATE    GROUP      INBOUND     OUTBOUND
192.168.10.10   up       ipmp0         net0    net0 net1
192.168.10.15   up       ipmp0         net1    net0 net1
192.0.0.100     up       acctg1        --          --
192.0.0.101     up       acctg1        --          --
192.168.10.31   up       field2        net2    net2 net7
192.168.10.32   up       field2        net7    net2 net7
192.168.10.33   down     field2        --          --
```

The output fields provide the following information:

`ADDRESS`	Specifies the host name or the data address, if the `-n` option is used with the `-a` option.
`STATE`	Indicates whether the address on the IPMP interface is `up`, and therefore usable, or `down`, and therefore unusable.

`GROUP`	Specifies the IPMP interface that hosts a specific data address. Typically, in Oracle Solaris, the name of the IPMP group is the IPMP interface.
`INBOUND`	Identifies the interface that receives packets for a given address. The field information might change depending on external events. For example, if a data address is down, or if no active IP interfaces remain in the IPMP group, this field is empty. The empty field indicates that the system is not accepting IP packets that are destined for the given address.
`OUTBOUND`	Identifies the interface that sends packets that are using a given address as a source address. As with the `INBOUND` field, the `OUTBOUND` information might also change depending on external events. An empty field indicates that the system is not sending packets with the given source address. The field might be empty, either because the address is down or because no active IP interfaces remain in the group.

EXAMPLE 27 Obtaining Information About Underlying IP Interfaces of an IPMP Group

The `-i` option displays information about an IPMP group's underlying IP interfaces.

```
% ipmpstat -i
INTERFACE   ACTIVE   GROUP      FLAGS      LINK       PROBE      STATE
net0        yes      ipmp0      --mb---    up         ok         ok
net1        yes      ipmp0      -------    up         disabled   ok
net3        no       acctg1     -------    unknown    disabled   offline
net4        no       acctg1     is-----    down       unknown    failed
net2        yes      field2     --mb---    unknown    ok         ok
net6        no       field2     -i-----    up         ok         ok
net5        no       filed2     -------    up         failed     failed
net7        yes      field2     --mb---    up         ok         ok
```

The output fields provide the following information:

`INTERFACE`	Specifies each underlying interface in each IPMP group.
`ACTIVE`	Indicates whether the interface is functioning and in use (`yes`) or not (`no`).
`GROUP`	Specifies the IPMP interface name. For anonymous groups, this field is empty. For more information about anonymous groups, see the `in.mpathd`(1M) man page.
`FLAGS`	Indicates the status of each underlying interface, which can be one or any combination of the following: ■ `b` – Indicates that the interface is designated by the system to receive broadcast traffic for the IPMP group. ■ `d` – Indicates that the interface is down and therefore unusable.

- `h` – Indicates that the interface shares a duplicate physical hardware address with another interface and has been taken offline. The `h` flag indicates that the interface is unusable.
- `i` – Indicates that the `INACTIVE` flag is set for the interface. Therefore, the interface is not used to send or receive data traffic.
- `m` – Indicates that the interface is designated by the system to send and receive IPv4 multicast traffic for the IPMP group.
- `M` – Indicates that the interface is designated by the system to send and receive IPv6 multicast traffic for the IPMP group.
- `s` – Indicates that the interface is configured as a standby interface.

`LINK`
: Indicates the status of link-based failure detection, which is one of the following:

- `up` or `down` – Indicates the availability or unavailability of a link.
- `unknown` – Indicates that the driver does not support notification of whether a link is `up` or `down` and therefore does not detect changes in the state of the link.

`PROBE`
: Specifies the state of probe-based failure detection for interfaces that have been configured with a test address, as follows:

- `ok` – Indicates that the probe is functional and active.
- `failed` – Indicates that probe-based failure detection has detected that the interface is not working.
- `unknown` – Indicates that no suitable probe targets could be found, Therefore, probes cannot be sent.
- `disabled` – Indicates that no IPMP test address is configured on the interface. Therefore, probe-based failure detection is disabled.

`STATE`
: Specifies the overall state of the interface, as follows:

- `ok` – Indicates that the interface is online and working normally based on the configuration of failure detection methods.
- `failed` – Indicates that the interface is not working either because the interface's link is down or because the probe detection has determined that the interface cannot send or receive traffic.
- `offline` – Indicates that the interface is not available for use. Typically, the interface is taken offline under the following circumstances:
  - The interface is being tested.
  - Dynamic reconfiguration is being performed.
  - The interface shares a duplicate hardware address with another interface.

- `unknown` – Indicates that the IPMP interface's state cannot be determined because no probe targets were found for probe-based failure detection.

EXAMPLE 28 Obtaining IPMP Probe Target Information

The `-t` option identifies the probe targets that are associated with each IP interface in an IPMP group. The output in the following example shows an IPMP configuration that uses test addresses for probe-based failure detection.

```
% ipmpstat -nt
INTERFACE   MODE          TESTADDR        TARGETS
net0        routes        192.168.85.30   192.168.85.1 192.168.85.3
net1        disabled      --              --
net3        disabled      --              --
net4        routes        192.1.2.200      192.1.2.1
net2        multicast     192.168.10.200   192.168.10.1 192.168.10.2
net6        multicast     192.168.10.201   192.168.10.2 192.168.10.1
net5        multicast     192.168.10.202   192.168.10.1 192.168.10.2
net7        multicast     192.168.10.203   192.168.10.1 192.168.10.2
```

The following output shows an IPMP configuration that uses transitive probing or probe-based failure detection without test addresses.

```
% ipmpstat -nt
INTERFACE   MODE         TESTADDR         TARGETS
net3        transitive   <net1>           <net1> <net2> <net3>
net2        transitive   <net1>           <net1> <net2> <net3>
net1        routes       172.16.30.100    172.16.30.1
```

The output fields provide the following information:

`INTERFACE`	Specifies each underlying interface of an IPMP group.
`MODE`	Specifies the method for obtaining the probe targets.

- `routes` – Indicates that the system routing table is used to find probe targets.
- `mcast` – Indicates that multicast ICMP probes are used to find targets.
- `disabled` – Indicates that probe-based failure detection has been disabled for the interface.
- `transitive` – Indicates that transitive probing is used for failure detection, as shown in the second example. Note that you cannot implement probe-based failure detection while simultaneously using transitive probes and test addresses. If you do not want to use test addresses, then you must enable transitive probing. If you do not want to use transitive probing, then you must

configure test addresses. For an overview, see "Probe-Based Failure Detection" on page 69.

`TESTADDR` Specifies the host name, or if the `-n` option is used with the `-t` option, the IP address that is assigned to the interface to send and receive probes.

If transitive probing is used, then the interface names refer to the underlying IP interfaces that are not actively used to receive data. The names also indicate that the transitive test probes are being sent with the source address of these specified interfaces. For active underlying IP interfaces that receive data, an IP address that is displayed indicates the source address of outgoing ICMP probes.

Note - If an IP interface is configured with both IPv4 and IPv6 test addresses, the probe target information is displayed separately for each test address.

`TARGETS` Lists the current probe targets in a space-separated list. The probe targets are displayed either as host names or IP addresses. If the `-n` option is used with the `-t` option, the IP addresses are displayed.

EXAMPLE 29 Observing IPMP Probes

The `-p` option enables you to observe ongoing probes. When you use this option with the `ipmpstat` command, information about probe activity on the system is continuously displayed until you terminate the command by pressing Control-C. You must become the `root` role or have appropriate privileges to run this command.

The following is an example of an IPMP configuration that uses test addresses for probe-based failure detection.

```
# ipmpstat -pn
TIME     INTERFACE    PROBE      NETRTT    RTT      RTTAVG     TARGET
0.11s    net0         589        0.51ms    0.76ms   0.76ms     192.168.85.1
0.17s    net4         612        --        --       --         192.1.2.1
0.25s    net2         602        0.61ms    1.10ms   1.10ms     192.168.10.1
0.26s    net6         602        --        --       --         192.168.10.2
0.25s    net5         601        0.62ms    1.20ms   1.00ms     192.168.10.1
0.26s    net7         603        0.79ms    1.11ms   1.10ms     192.168.10.1
1.66s    net4         613        --        --       --         192.1.2.1
1.70s    net0         603        0.63ms    1.10ms   1.10ms     192.168.85.3
^C
```

The following is an example of an IPMP configuration that uses transitive probing or probe-based failure detection without test addresses.

```
# ipmpstat -pn
TIME     INTERFACE    PROBE       NETRTT    RTT      RTTAVG     TARGET
1.39S    net4         t28         1.05ms    1.06ms   1.15ms     <net1>
```

```
1.39s   net1        i29       1.00ms   1.42ms   1.48ms     172.16.30.1
^C
```

The output fields provide the following information:

`TIME`	Specifies the time a probe was sent relative to when the `ipmpstat` command was issued. If a probe was initiated prior to `ipmpstat` being started, then the time is displayed with a negative value, relative to when the command was issued.
`INTERFACE`	Specifies the interface on which the probe is sent.
`PROBE`	Specifies the identifier that represents the probe. If transitive probing is used for failure detection, the identifier is prefixed with either `t` for transitive probes or `i` for ICMP probes.
`NETRTT`	Specifies the total network round-trip time of the probe, measured in milliseconds. `NETRTT` covers the time between the moment when the IP module sends the probe and the moment the IP module receives the `ack` packets from the target. If the `in.mpathd` daemon has determined that the probe is lost, then the field is empty.
`RTT`	Specifies the total round-trip time for the probe, measured in milliseconds. `RTT` covers the time between the moment the `in.mpathd` daemon executes the code to send the probe and the moment the daemon completes processing of the `ack` packets from the target. If the daemon has determined that the probe is lost, then the field is empty. Spikes that occur in the `RTT` that are not present in the `NETRTT` might indicate that the local system is overloaded.
`RTTAVG`	Specifies the probe's average round-trip time over the interface between the local system and the target. The average round-trip time helps identify slow targets. If data is insufficient to calculate the average, this field is empty.
`TARGET`	Specifies the host name. Or, if the `-n` option is used with the `-p` option, specifies the target address to which the probe is sent.

Customizing the Output of the `ipmpstat` Command

The `-o` option enables you to customize the output of the `ipmpstat` command. You use this option with the other previously mentioned `ipmpstat` options to select specific fields to be displayed out of the total fields that the main option normally displays.

For example, the `-g` option provides the following information:

- IPMP group
- IPMP group name
- Status of the group
- Failure detection time
- Underlying interfaces of the IPMP group

Suppose that you want to display only the status of the IPMP groups on the system. You would combine the `-o` and `-g` options and specify the `groupname` and `state` fields, as shown in the following example:

```
% ipmpstat -g -o groupname,state
GROUPNAME  STATE
ipmp0      ok
accgt1     failed
field2     degraded
```

To display all of the fields of the `ipmpstat` command for a specific type of information, include the `-o` option with the `all` argument.

Using the `ipmpstat` Command in Scripts

The `-o` option is useful when you run the `ipmpstat` command from a script or by using a command alias, particularly if you also want to generate machine-parsable output.

To generate machine-parsable information, you combine the `-P` and `-o` options with one of the other main `ipmpstat` options, along with the specific fields that you want to display.

A machine-parsable output differs from normal output in the following ways:

- Column headers are omitted.
- Fields are separated by colons (`:`).
- Fields with empty values are empty rather than filled with the double dash (`--`).
- When multiple fields are requested, if a field contains a literal colon (`:`) or backslash (`\`), you can escape or exclude these characters by prefixing them with a backslash (`\`).

To correctly use the `ipmpstat -P` command, observe the following rules:

- Use the `-o` *option field* option with the `-P` option. Separate multiple option fields with commas.
- Never use the `-o all` option with the `-P` option.

Caution - Ignoring either one of these rules causes `ipmpstat -P` to fail.

The following example shows the correct syntax for using the `-P` option:

```
% ipmpstat -P -o -g groupname,fdt,interfaces
ipmp0:10.00s:net0 net1
acctg1::[net3 net4]
field2:20.00s:net2 net7 (net5) [net6]
```

The group name, failure detection time, and underlying interfaces are group information fields. Thus, you use the `-o` and `-g` options along with the `-P` option.

The `-P` option is intended for use in scripts. The following example shows how you would run the `ipmpstat` command from a script. The script displays the failure detection time for an IPMP group.

```
getfdt() {
ipmpstat -gP -o group,fdt | while IFS=: read group fdt; do
[[ "$group" = "$1" ]] && { echo "$fdt"; return; }
done
}
```

♦ ♦ ♦ C H A P T E R 4

About IP Tunnel Administration

This chapter provides an overview of IP tunnel administration in Oracle Solaris. For task-related information, see Chapter 5, "Administering IP Tunnels".

This chapter contains the following topics:

- "What's New in IP Tunnel Administration" on page 105
- "About the IP Tunnel Feature" on page 105
- "About Deploying IP Tunnels" on page 112

What's New in IP Tunnel Administration

IP tunnel administration in Oracle Solaris 11 has been revised so that it is consistent with the new model that is used for network datalink administration. In this release, you create and configure IP tunnels by using the `dladm` command. Tunnels can also use other datalink features that are supported in this release. For example, support for administratively chosen names enables you to assign tunnels more meaningful names than in previous releases. For more information, see the `dladm`(1M) man page.

About the IP Tunnel Feature

IP tunnels, also referred to simply as tunnels in this book, provide a means for transporting data packets between domains when the protocol in those domains is not supported by intermediary networks. For example, IPv6 networks require a way to communicate outside their borders in an environment where most networks use the IPv4 protocol. This communication is possible by using tunnels. IP tunnels provide a virtual link between two nodes that are reachable by using IP. The link can thus be used to transport IPv6 packets over the IPv4 networks to enable IPv6 communication between the two IPv6 sites.

Types of Tunnels

Tunneling involves the encapsulation of an IP packet within another packet. This encapsulation enables the packet to reach its destination through intermediary networks that do not support the packet's protocol. Tunnels differ depending on the type of packet encapsulation that is used.

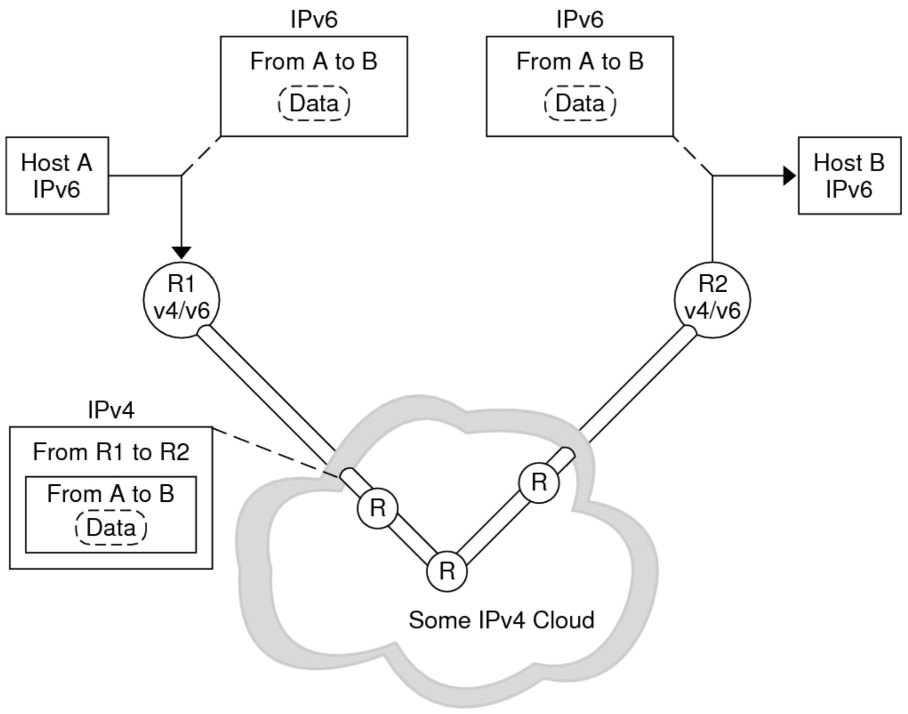
The following types of tunnels are supported in Oracle Solaris:

- **IPv4 tunnels** – IPv4 packets are encapsulated in an IPv4 header and sent to a preconfigured unicast IPv4 destination. To indicate more specifically the packets that flow over the tunnel, IPv4 tunnels are also called either *IPv4 over IPv4 tunnels* or *IPv6 over IPv4 tunnels*.
- **6to4 tunnels** – IPv6 packets are encapsulated in an IPv4 header and sent to an IPv4 destination that is automatically determined on a per-packet basis. This determination is based on an algorithm that is defined in the *6to4 protocol*.
- **IPv6 tunnels** – IPv6 packets are encapsulated in an IPv4 header and sent to an IPv4 destination that is automatically determined on a per-packet basis. The determination is based on an algorithm that is defined in the 6to4 protocol.

Tunnels in the Combined IPv6 and IPv4 Network Environments

Many sites that have IPv6 domains might need to communicate with other IPv6 domains by traversing IPv4 networks during the early phases of IPv6 deployment. The following figure illustrates the tunneling mechanism (indicated by "R" in the figure) between two IPv6 hosts through IPv4 routers.

FIGURE 5 IPv6 Tunneling Mechanism



In the previous figure, the tunnel consists of two routers that are configured with a virtual point-to-point link between the two routers over the IPv4 network.

An IPv6 packet is encapsulated within an IPv4 packet. The boundary router of the IPv6 network sets up a point-to-point tunnel over various IPv4 networks to the boundary router of the destination IPv6 network. The packet is transported over the tunnel to the destination boundary router, where the packet is decapsulated. The router then forwards the separate IPv6 packet to the destination node.

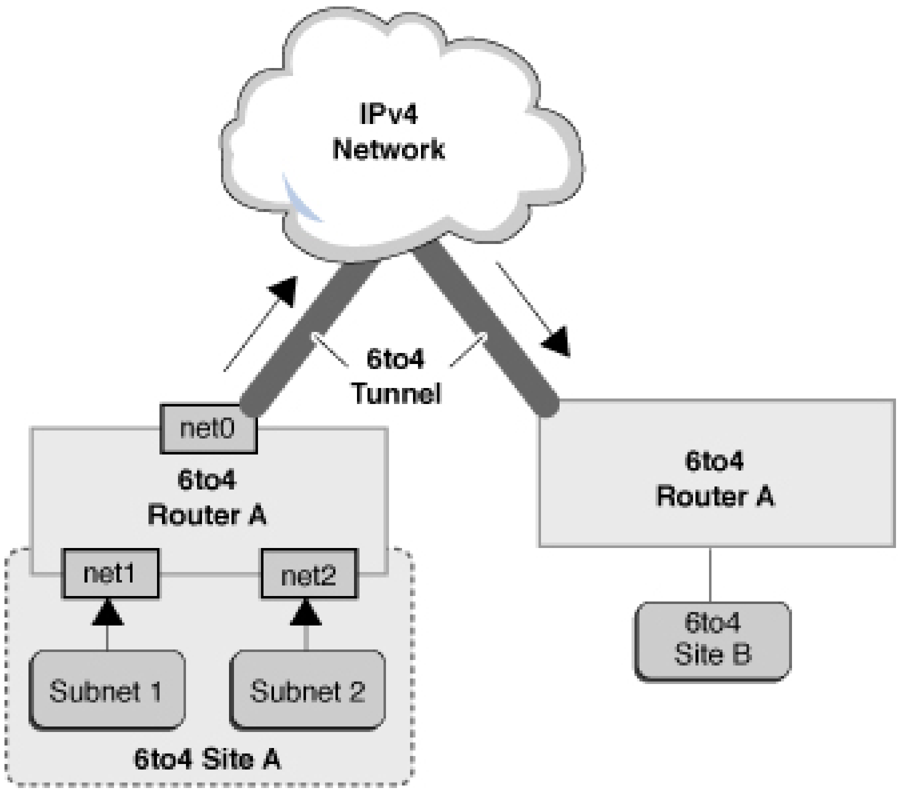
6to4 Tunnels

Oracle Solaris includes 6to4 tunnels as an interim method for making the transition from IPv4 to IPv6 addressing. *6to4 tunnels* enable isolated IPv6 sites to communicate across an automatic tunnel over an IPv4 network that does not support IPv6. To use 6to4 tunnels, you must first configure a boundary router on your IPv6 network as one endpoint of the 6to4 automatic tunnel. In this way, the 6to4 router can participate in a tunnel to another 6to4 site or to a native IPv6 non-6to4 site, if required.

Topology of a 6to4 Tunnel

A 6to4 tunnel provides IPv6 connectivity to all 6to4 sites everywhere. Likewise, the tunnel also functions as a link to all IPv6 sites, including the native IPv6 Internet, provided that the tunnel is configured to forward to a relay router. The following figure shows how a 6to4 tunnel provides this connectivity between 6to4 sites.

FIGURE 6 Tunnel Between Two 6to4 Sites



The previous figure depicts two isolated 6to4 networks, Site A and Site B. Each site has configured a router with an external connection to an IPv4 network. A 6to4 tunnel across the IPv4 network provides a connection to link 6to4 sites.

Prior to an IPv6 site becoming a 6to4 site, you must configure at least one router interface for 6to4 support. This interface must provide the external connection to the IPv4 network. In the previous figure, boundary Router A's interface `net0` connects Site A to the IPv4 network. The address that you configure on `net0` must be globally unique. You must configure the `net0` interface with an IPv4 address before you can configure a tunnel interface for 6to4 support on the router.

In the figure, 6to4 Site A is composed of two subnets that are connected to interfaces `net1` and `net2` on Router A. All IPv6 hosts on either subnet of Site A are automatically reconfigured with 6to4-derived addresses upon receipt of the advertisement from Router A.

Site B is another isolated 6to4 site. To correctly receive traffic from Site A, you must configure a boundary router on Site B for 6to4 support. Otherwise, packets that the router receives from Site A are not recognized and are then dropped.

`6to4relay` Command

6to4 tunneling enables communication between isolated 6to4 sites. However, to transfer packets with a native, non-6to4 IPv6 site, the 6to4 router must establish a tunnel with a 6to4 relay router. The *6to4 relay router* then forwards the 6to4 packets to the IPv6 network, and ultimately, to the native IPv6 site. If your 6to4-enabled site must exchange data with a native IPv6 site, you use the `6to4relay` command to enable the appropriate tunnel.

Note - Because the use of relay routers is insecure, tunneling to a relay router is disabled by default in Oracle Solaris. Carefully consider the issues that are involved in creating a tunnel to a 6to4 relay router before deploying this scenario. For detailed information, see "Considerations for Enabling Tunnels to a 6to4 Relay Router" on page 110. If you decide to enable 6to4 relay router support, you can find the related procedures in "How to Create and Configure an IP Tunnel" on page 116.

For more information, see the `6to4`(7M) man page.

Packet Flow Through a 6to4 Tunnel

The following information pertains to how the flow of packets from a system at one 6to4 site to a system at a remote 6to4 site works. This scenario that is described uses the topology that is shown in Figure 6, "Tunnel Between Two 6to4 Sites," on page 108. Moreover, the scenario assumes that the 6to4 routers and the 6to4 hosts are already configured.

The packet flow is as follows:

1. A system on Subnet 1 of 6to4 Site A sends a transmission with a system at 6to4 Site B as the destination. Each packet header has a 6to4-derived source address and a 6to4-derived destination address.
2. Site A's router encapsulates each 6to4 packet within an IPv4 header. In this process, the router sets the IPv4 destination address of the encapsulating header to Site B's router

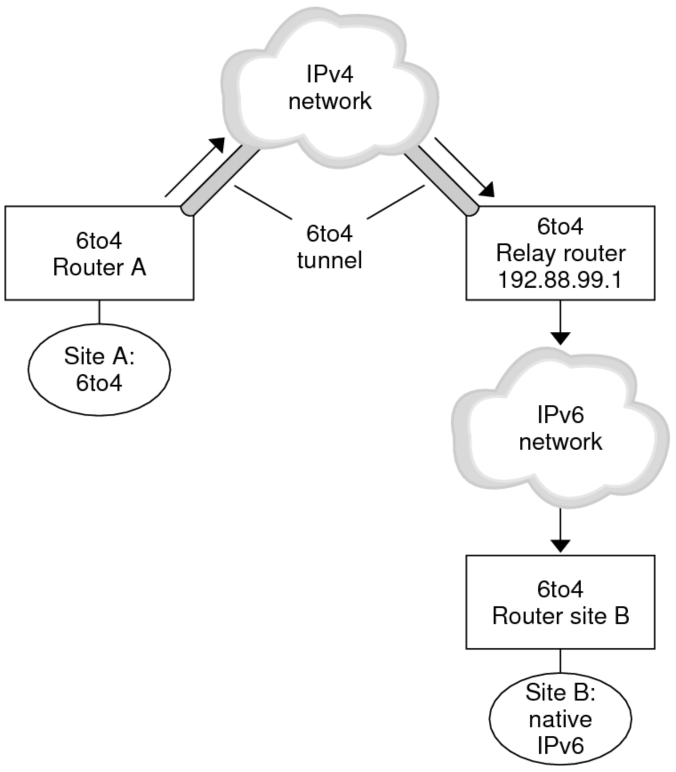
address. For each IPv6 packet that flows through the tunnel interface, the packet's IPv6 destination address also contains the IPv4 destination address. Thus, the router is able to determine the IPv4 destination address that is set on the encapsulating header. Then, the router uses standard IPv4 routing procedures to forward the packet over the IPv4 network.

3. Any IPv4 routers that the packets encounter use the packets' IPv4 destination address for forwarding. This address is the globally unique IPv4 address of the interface on Router B, which also serves as the 6to4 pseudo-interface.
4. Packets from Site A arrive at Router B, which decapsulates the IPv6 packets from the IPv4 header.
5. Router B then uses the destination address in the IPv6 packet to forward the packets to the recipient system at Site B.

Considerations for Enabling Tunnels to a 6to4 Relay Router

6to4 relay routers function as endpoints for tunnels from 6to4 routers that need to communicate with native IPv6, non-6to4 networks. Relay routers are essentially bridges between the 6to4 site and native IPv6 sites. Because this solution might be insecure, by default, Oracle Solaris does not enable 6to4 relay router support. However, if your site requires such a tunnel, you can use the `6to4relay` command to enable tunneling, as depicted in the following figure.

FIGURE 7 Tunnel From a 6to4 Site to a 6to4 Relay Router



In Figure 7, "Tunnel From a 6to4 Site to a 6to4 Relay Router," on page 111, 6to4 Site A needs to communicate with a node at the native IPv6 Site B. The figure shows the path of traffic from Site A onto a 6to4 tunnel over an IPv4 network. The tunnel has 6to4 Router A and a 6to4 relay router as its endpoints. Beyond the 6to4 relay router is the IPv6 network, to which IPv6 Site B is connected.

Packet Flow Between a 6to4 Site and a Native IPv6 Site

The following information pertains to the flow of packets from a 6to4 site to a native IPv6 site. This scenario uses the topology that is shown in Figure 7, "Tunnel From a 6to4 Site to a 6to4 Relay Router," on page 111.

The packet flow is as follows:

1. A system on 6to4 Site A sends a transmission that specifies as the destination a system at native IPv6 Site B. Each packet header has a 6to4-derived address as its source address. The destination address is a standard IPv6 address.
2. Site A's 6to4 router encapsulates each packet within an IPv4 header, which has the IPv4 address of the 6to4 relay router as its destination. The 6to4 router uses standard IPv4 routing procedures to forward the packet over the IPv4 network. Any IPv4 routers that the packets encounter forward the packets to the 6to4 relay router.
3. The physically closest anycast 6to4 relay router to Site A retrieves the packets that are destined for the `192.88.99.1` anycast group.

Note - 6to4 relay routers that are part of the 6to4 relay router anycast group have the IP address `192.88.99.1`. This anycast address is the default address for 6to4 relay routers. If you need to use a specific 6to4 relay router, you can override the default and specify that router's IPv4 address.

4. The relay router decapsulates the IPv4 header from the 6to4 packets, revealing the native IPv6 destination address.
5. The relay router then sends the now IPv6-only packets onto the IPv6 network, where the packets are ultimately retrieved by a router at Site B. The router then forwards the packets to the destination IPv6 node.

About Deploying IP Tunnels

To properly deploy IP tunnels, you need to perform two main tasks. First, create the tunnel link, then configure an IP interface over the tunnel. The following are the requirements for creating tunnels and their corresponding IP interfaces.

Requirements for Creating IP Tunnels

To successfully create IP tunnels, you must observe the following requirements:

- If you use host names instead of literal IP addresses, these names must resolve to valid IP addresses that are compatible with the tunnel type.
- The IPv4 or IPv6 tunnel that you create must not share the same tunnel source address and tunnel destination address with another configured tunnel.
- The IPv4 or IPv6 tunnel that you create must not share the same tunnel source address with an existing 6to4 tunnel.
- If you create a 6to4 tunnel, that tunnel must not share the same tunnel source address with another configured tunnel.

For more information, see "Planning for Tunnel Use on the Network" in *Planning for Network Deployment in Oracle Solaris 11.3*.

Requirements for IP Tunnels and IP Interfaces

Each tunnel type has specific IP address requirements for the IP interface that you configure over the tunnel. These requirements are summarized in the following table.

TABLE 2 Tunnels and IP Interface Requirements

Tunnel Type	IP Interface Allowed Over Tunnel	IP Interface Requirement
IPv4 tunnel	IPv4 interface	Local and remote addresses are manually specified.
IPv4 tunnel	IPv6 interface	Local and remote link-local addresses are automatically set when you issue the `ipadm create-addr -T addrconf` command. See the `ipadm`(1M) man page.
IPv6 tunnel	IPv4 interface	Local and remote addresses are manually specified.
IPv6 tunnel	IPv6 interface	Local and remote link-local addresses are automatically set when you issue the `ipadm create-addr -T addrconf` command. See the `ipadm`(1M) man page.
6to4 tunnel	IPv6 interface only	Default IPv6 address is automatically selected when you issue the `ipadm create-ip` command. See the `ipadm`(1M) man page.

You can override the link-local addresses that are automatically set for IPv6 interfaces over IPv6 or IPv4 tunnels by specifying a local and remote `interface-id` with the `ipadm create-addr -T addrconf` command.

♦ ♦ ♦ C H A P T E R 5

Administering IP Tunnels

This chapter describes tasks for administering IP tunnels in Oracle Solaris. For an overview of IP tunnel administration, see Chapter 4, "About IP Tunnel Administration".

This chapter contains the following topics:

- "About IP Tunnel Administration in Oracle Solaris" on page 115
- "Administering IP Tunnels" on page 116

About IP Tunnel Administration in Oracle Solaris

In this Oracle Solaris release, tunnel administration is separated from IP interface configuration. You administer the datalink aspect of IP tunnels with the `dladm` command and the IP aspect of configuration (including those for IP tunnels) by using the `ipadm` command.

The following `dladm` subcommands are used to configure IP tunnels:

- `create-iptun`
- `modify-iptun`
- `show-iptun`
- `delete-iptun`
- `set-linkprop`

For more information, see the `dladm`(1M) man page.

IP tunnel administration is closely associated with IPsec configuration. For example, IPsec virtual private networks (VPNs) are one of the primary uses of IP tunneling. For more information about network security in Oracle Solaris, see Chapter 8, "About IP Security Architecture" in *Securing the Network in Oracle Solaris 11.3*. To configure IPsec, see Chapter 9, "Configuring IPsec" in *Securing the Network in Oracle Solaris 11.3*.

Administering IP Tunnels

This section contains the following topics:

▼ How to Create and Configure an IP Tunnel

1. **Become the `root` role.**

2. **Create the tunnel.**

```
# dladm create-iptun [-t] -T type -a [local|remote]=addr,... tunnel-link
```

`-t`	Creates a temporary tunnel. By default, the command creates a persistent tunnel. If you want to configure a persistent IP interface over the tunnel, then you must create a persistent tunnel and not use the `-t` option.
`-T` *type*	Specifies the type of tunnel you want to create (IPv4 or IPv6). This argument is required to create all tunnel types.
`-a [local|remote]=`*address,...*	Specifies literal IP addresses or host names that correspond to the local address and the remote tunnel address. The addresses must be valid and already created in the system. Depending on the type of tunnel, you specify either only one address, or both local and remote addresses. If specifying both local and remote addresses, you must separate the addresses with a comma. ■ `IPv4` tunnels require local and remote IPv4 addresses to function. ■ `IPv6` tunnels require local and remote IPv6 addresses to function. ■ `6to4` tunnels require a local IPv4 address to function.

Note - For persistent IP tunnel datalink configurations, if you are using host names for addresses, these host names are saved in the configuration storage. During a subsequent system boot, if the names resolve to IP addresses that are different from the IP addresses used when the tunnel was created, then the tunnel acquires a new configuration.

---

tunnel-link	Specifies the IP tunnel link. With support for meaningful names in a datalink administration, tunnel names are no longer restricted to the type of tunnel that you are creating. Instead, you can assign any administratively chosen name to a tunnel. Tunnel names consist of a string and the physical point of attachment (PPA) number, for example, *mytunnel0*. For rules governing the assignment of meaningful names, refer to "Rules for Valid Link Names" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.

3. **(Optional) Set values for the hop limit or the encapsulation limit.**

```
# dladm set-linkprop -p [hoplimit=value] [encaplimit=value] tunnel-link
```

`hoplimit`	Specifies the hop limit of the tunnel interface for tunneling over IPv6. The *hoplimit* is the equivalent of the IPv4 time to live (TTL) field for tunneling over IPv4.
`encaplimit`	Specifies the number of levels of nested tunneling that are allowed for a packet. This option applies only to IPv6 tunnels.

The values that you set for the `hoplimit` and `encaplimit` properties must remain within acceptable ranges. The `hoplimit` and `encaplimit` properties are tunnel link properties. Thus, these properties are administered by the same `dladm` subcommands as other link properties. The subcommands that you use are `dladm set-linkprop`, `dladm reset-linkprop`, and `dladm show-linkprop`.

4. **Create an IP interface over the tunnel.**

```
# ipadm create-ip tunnel-interface
```

where *tunnel-interface* uses the same name as the tunnel link.

5. **Assign local and remote IP addresses to the tunnel interface.**

```
# ipadm create-addr [-t] -a local=address,remote=address  interface
```

where *interface* specifies the tunnel interface.

For more information, see the `ipadm(1M)` man page and *Configuring and Managing Network Components in Oracle Solaris 11.3*.

6. **(Optional) Verify the status of the tunnel's IP interface configuration.**

```
# ipadm show-addr interface
```

Example 30 Creating an IPv6 Interface Over an IPv4 Tunnel

The following example shows how you would create a persistent IPv6 over IPv4 tunnel.

```
# dladm create-iptun -T ipv4 -a local=192.0.2.23,remote=203.0.113.14 private0
# dladm set-linkprop -p hoplimit=200 private0
# ipadm create-ip private0
# ipadm create-addr -T addrconf private0
private0/v6
# ipadm show-addr private0/
ADDROBJ           TYPE     STATE        ADDR
private0/v6       addrconf ok fe80::c000:217->fe80::cb00:710e
```

To add alternative addresses, use the same syntax. For example, you can add a global address as follows:

```
# ipadm create-addr -a local=2001:db8:4728::1,remote=2001:db8:4728::2 private0
private0/v6a
# ipadm show-addr private0/
ADDROBJ           TYPE     STATE        ADDR
private0/v6       addrconf ok fe80::c000:217->fe80::cb00:710e
private0/v6a      static   ok 2001:db8:4728::1->2001:db8:4728::2
```

Note that the prefix `2001:db8` for the IPv6 address is a special IPv6 prefix that is used specifically for documentation examples.

Example 31 Creating an IPv4 Interface Over an IPv4 Tunnel

The following example shows how you would create a persistent IPv4 over IPv4 tunnel.

```
# dladm create-iptun -T ipv4 -a local=192.0.2.23,remote=203.0.113.14 vpn0
# ipadm create-ip vpn0
# ipadm create-addr -a local=10.0.0.1,remote=10.0.0.2 vpn0
vpn0/v4
# ipadm show-addr vpn0/
ADDROBJ           TYPE     STATE        ADDR
vpn0/v4           static   ok 10.0.0.1->10.0.0.2
```

You can further configure IPsec policy to provide secure connections for the packets that flow over this tunnel. For information, see Chapter 9, "Configuring IPsec" in *Securing the Network in Oracle Solaris 11.3*.

Example 32 Creating an IPv6 Interface Over an IPv6 Tunnel

The following example shows how you would create a persistent IPv6 over IPv6 tunnel.

```
# dladm create-iptun -T ipv6 -a local=2001:db8:feed::1234,remote=2001:db8:beef::4321
 tun0
# ipadm create-ip tun0
# ipadm create-addr -T addrconf tun0
tun0/v6
# ipadm show-addr tun0/
ADDROBJ           TYPE     STATE        ADDR
tun0/v6           addrconf ok fe80::1234->fe80::4321
```

To add addresses, for example, a global address or alternative local and remote addresses, use the `ipadm` command as follows:

```
# ipadm create-addr -a local=2001:db8:cafe::1,remote=2001:db8:cafe::2 tun0
tun0/v6a
# ipadm show-addr tun0/
ADDROBJ           TYPE     STATE        ADDR
tun0/v6           addrconf ok fe80::1234->fe80::4321
tun0/v6a          static   ok 2001:db8:cafe::1->2001:db8:cafe::2
```

▼ How to Configure a 6to4 Tunnel

When configuring 6to4 tunnels, a 6to4 router must act as the IPv6 router to the nodes that are in the network's 6to4 sites. Thus, when configuring a 6to4 router, you must also configure the router as an IPv6 router on its physical interfaces. For more information about configuring an Oracle Solaris system as a router, see "Configuring an IPv6 Router" in *Configuring an Oracle Solaris 11.3 System as a Router or a Load Balancer*.

1. **Create a 6to4 tunnel.**

   ```
   # dladm create-iptun -T 6to4 -a local=address tunnel-link
   ```

-a local=*address*	Specifies the tunnel local address, which must already be existing in the system to be a valid address.
tunnel-link	Specifies the IP tunnel link that you can assign a tunnel.

2. **Create the tunnel IP interface.**

   ```
   # ipadm create-ip tunnel-interface
   ```

   where *tunnel-interface* uses the same name as the tunnel link.

   The system automatically configures an IPv6 address for the local interface by using interface-id `::1`.

3. **(Optional) Add alternative IPv6 addresses for the tunnel's use.**

4. **Edit the `/etc/inet/ndpd.conf` file.**

```
# pfedit /etc/inet/ndpd.conf
```

5. **Advertise 6to4 routing by adding the following two lines to the file.**

   ```
   if subnet-interface AdvSendAdvertisements 1
   prefix IPv6-prefix subnet-interface
   ```

   where the first line specifies the local IPv6 interface to send router advertisements over and the second line specifies the IPv6 subnet prefix to use on the LAN that is attached to that interface. The IPv6 prefix must start with the same 48-bit 6to4 prefix that is used on the 6to4 tunnel interface.

   For detailed information about the `ndpd.conf` file, see the `ndpd.conf(4)` man page.

6. **Enable IPv6 forwarding.**

   ```
   # ipadm set-prop -p forwarding=on ipv6
   ```

7. **Choose from one of the following options:**

   - **Reboot the router.**
   - **Issue a `sighup` to the `/etc/inet/in.ndpd` daemon to begin sending router advertisements.**

     The IPv6 nodes on each subnet to receive the 6to4 prefix autoconfigured with the new 6to4-derived addresses.

8. **Add the new 6to4-derived addresses for all of the nodes in the 6to4 site to the name service database.**

   For instructions, see Chapter 4, "Administering Naming and Directory Services on an Oracle Solaris Client" in *Configuring and Managing Network Components in Oracle Solaris 11.3*.

Example 33 Creating a 6to4 Tunnel

The following example shows you would create a 6to4 tunnel. Note that only IPv6 interfaces can be configured over 6to4 tunnels. In this example, the subnet interface is `net0` to which the `/etc/inet/ndpd.conf` refers.

```
# dladm create-iptun -T 6to4 -a local=192.0.2.23 tun0
# ipadm create-ip tun0
# ipadm show-addr
ADDROBJ           TYPE     STATE        ADDR
lo0/v4            static   ok           127.0.0.1/8
net0/v4           dhcp     ok           192.0.2.23/24
lo0/v6            static   ok           ::1/128
tun0/v6           static   ok           2002:c000:217::1/16
```

```
# ipadm create-addr -T addrconf net0
net0/v6
# ipadm create-addr -a 2002:c000:217:cafe::1 net0
net0/v6a
# ipadm show-addr
ADDROBJ           TYPE     STATE        ADDR
lo0/v4            static   ok           127.0.0.1/8
net0/v4           dhcp     ok           192.0.2.23/24
lo0/v6            static   ok           ::1/128
net0/v6           addrconf ok      fe80::214:4fff:fef9:b1a9/10
net0/v6a          static   ok           2002:c000:217:cafe::1/64
tun0/v6           static   ok           2002:c000:217::1/16

# vi /etc/inet/ndpd.conf
if net0 AdvSendAdvertisements on
prefix 2002:c000:217:cafe::0/64 net0

# ipadm set-prop -p forwarding=on ipv6
```

Note that for 6to4 tunnels, the prefix for the IPv6 address is `2002`.

▼ How to Enable a 6to4 Tunnel to a 6to4 Relay Router

Caution - Due to major security issues, 6to4 relay router support is disabled in Oracle Solaris by default. See "Security Issues When Tunneling to a 6to4 Relay Router" in *Troubleshooting Network Administration Issues in Oracle Solaris 11.3* for details.

Before You Begin Before you enable a 6to4 tunnel to a 6to4 relay router, complete the following tasks:

- Configure a 6to4 router at your site. See "How to Create and Configure an IP Tunnel" on page 116.
- Review the security issues that are involved in tunneling to a 6to4 relay router.

1. **Enable a tunnel to the 6to4 relay router by using either of the following methods:**

   - **Enable a tunnel to an anycast 6to4 relay router.**

     ```
     # 6to4relay -e
     ```

     The `-e` option sets up a tunnel between the 6to4 router and an anycast 6to4 relay router. Anycast 6to4 relay routers have the well-known IPv4 address `192.88.99.1`. The anycast relay router that is physically nearest to your site becomes the endpoint for the 6to4 tunnel. This relay router then handles packet forwarding between your 6to4 site and a native IPv6 site.

For detailed information, refer to RFC 3068, "An Anycast Prefix for 6to4 Relay Routers" (`http://www.rfc-editor.org/rfc/rfc3068.txt`).

- **Enable a tunnel to a specific 6to4 relay router.**

  ```
  # 6to4relay -e -a relay-router-address
  ```

  The `-a` option indicates that a specific router address is to follow. Replace *relay-router-address* with the IPv4 address of the specific 6to4 relay router with which you want to enable a tunnel.

The tunnel to the 6to4 relay router remains active until you remove the 6to4 tunnel pseudo-interface.

2. **Delete the tunnel to the 6to4 relay router, when the tunnel is no longer needed.**

   ```
   # 6to4relay -d
   ```

3. **(Optional) Make the tunnel to the 6to4 relay router persistent across reboots.**
   Your site might have a compelling reason to have the tunnel to the 6to4 relay router reinstated each time the 6to4 router reboots. To support this scenario, you must do the following:

   a. **Edit the `/etc/default/inetinit` file.**

      ```
      # pfedit /etc/default/inetinit
      ```

      The line to modify is at the end of the file.

   b. **Change the `NO` value in the `ACCEPT6TO4RELAY=NO` line to `YES`.**

   c. **(Optional) Create a tunnel to a specific 6to4 relay router that persists across reboots.**
      For the parameter `RELAY6TO4ADDR`, change the address `192.88.99.1` to the IPv4 address of the 6to4 relay router that you want to use.

Example 34 Getting Status Information About 6to4 Relay Router Support

Use the `6to4relay` command to find out whether support for 6to4 relay routers is enabled. The following example shows the output when support for 6to4 relay routers is disabled, as is the default in Oracle Solaris.

```
# 6to4relay
6to4relay: 6to4 Relay Router communication support is disabled.
```

When support for 6to4 relay routers is enabled, the following output is displayed:

```
# 6to4relay
6to4relay: 6to4 Relay Router communication support is enabled.
```

```
IPv4 remote address of Relay Router=192.88.99.1
```

Modifying an IP Tunnel Configuration

You change a tunnel's configuration by using the following command syntax:

```
# dladm modify-iptun -a [local|remote]=addr,... tunnel-link
```

You cannot modify an existing tunnel's type. Thus, the `-T` *type* option is not allowed for this command. Only the following tunnel parameters can be modified:

`-a [local|remote]=`*address,...*

Specifies literal IP addresses or host names that correspond to the local address and the remote tunnel address. Depending on the type of tunnel, you specify either only one address, or both local and remote addresses. If you are specifying both local and remote addresses, you must separate the addresses with a comma.

- `IPv4` tunnels require local and remote IPv4 addresses to function.
- `IPv6` tunnels require local and remote IPv6 addresses to function.
- `6to4` tunnels require a local IPv4 address to function.

For persistent IP tunnel datalink configurations, if you are using host names for addresses, these host names are saved in the configuration storage. During a subsequent system boot, if the names resolve to IP addresses that are different from the IP addresses that were used when the tunnel was created, then the tunnel acquires a new configuration.

If you are changing the tunnel's local and remote addresses, ensure that these addresses are consistent with the type of tunnel that you are modifying.

- To change the name of the tunnel link, use the `dladm rename-link` command rather than the `modify-iptun` command as follows:

  ```
  # dladm rename-link old-tunnel-link new-tunnel-link
  ```
- To change tunnel properties such as the `hoplimit` or `encaplimit`, use the `dladm set-linkprop` command rather than the `modify-iptun` command.

EXAMPLE 35 Modifying a Tunnel's Address and Properties

The following example consists of two steps. The first command shows how to temporarily change the local and remote addresses of the IPv4 tunnel `vpn0`. Then, when the system is rebooted, the tunnel reverts to using the original addresses. The second command shows how to change the `hoplimit` of `vpn0` to 60.

```
# dladm modify-iptun -t -a local=10.8.48.149,remote=192.168.2.3 vpn0
```

```
# dladm set-linkprop -p hoplimit=60 vpn0
```

Displaying IP Tunnel Configuration Information

You display an IP tunnel's configuration by using the following command syntax:

```
# dladm show-iptun [-p] -o fields [tunnel-link]
```

`-p`	Displays the information in a machine-parsable format. This argument is optional.
`-o` *fields*	Displays selected fields that provide specific tunnel information.
tunnel-link	Specifies the tunnel whose configuration information you want to display. This argument is optional. If you omit the tunnel name, the command displays the information about all of the tunnels on in the system.

EXAMPLE 36 Displaying Information About All Tunnels

In the following example, only one tunnel exists on the system.

```
# dladm show-iptun
LINK     TYPE     FLAGS     LOCAL          REMOTE
tun0     6to4     --       192.168.35.10   --
vpn0     ipv4     --       10.8.48.149     192.168.2.3
```

EXAMPLE 37 Displaying Selected Fields in a Machine-Parsable Format

In the following example, only the specific fields with tunnel information are displayed.

```
# dladm show-iptun -p -o link,type,local
tun0:6to4:192.168.35.10
vpn0:ipv4:10.8.48.149
```

Displaying an IP Tunnel's Properties

You display tunnel link's properties by using the following command syntax:

```
# dladm show-linkprop [-c] [-o fields] [tunnel-link]
```

`-c`	Displays the information in a machine-parsable format. This argument is optional.
`-o` *fields*	Displays selected fields that provide specific information about the link's properties.

tunnel-link	Specifies the tunnel whose properties you want to display. This argument is optional. If you omit the tunnel name, the command displays the information about all of the tunnels on in the system.

EXAMPLE 38 Displaying a Tunnel's Properties

The following example shows how you would display all of a tunnel's link properties.

```
# dladm show-linkprop iptun0
LINK     PROPERTY        PERM VALUE        EFFECTIVE    DEFAULT   POSSIBLE
iptun0   autopush        rw   --           --           --        --
iptun0   zone            rw   --           --           --        --
iptun0   state           r-   up           up           up        up,down
iptun0   mtu             rw   1480         1480         1480      1280-1480
iptun0   maxbw           rw   --           --           --        --
iptun0   cpus            rw   --           --           --        --
iptun0   rxfanout        rw   --           8            8         --
iptun0   pool            rw   --           --           --        --
iptun0   priority        rw   medium       medium       medium    low,medium,
                                                                  high
iptun0   hoplimit        rw   64           64           64        1-255
iptun0   protection      rw   --           --           --        mac-nospoof,
                                                                  restricted,
                                                                  ip-nospoof,
                                                                  dhcp-nospoof
iptun0   allowed-ips     rw   --           --           --        --
iptun0   allowed-dhcp-cids rw --           --           --        --
iptun0   rxrings         rw   --           --           --        --
iptun0   txrings         rw   --           --           --        --
iptun0   txringsavail    r-   0            0            --        --
iptun0   rxringsavail    r-   0            0            --        --
iptun0   rxhwclntavail   r-   0            0            --        --
iptun0   txhwclntavail   r-   0            0            --        --
```

▼ How to Delete an IP Tunnel

1. **Unplumb the IP interface that is configured over the tunnel depending on the type of interface.**

   ```
   # ipadm delete-ip tunnel-link
   ```

   Note - To successfully delete a tunnel, no existing IP interface can be plumbed on the tunnel.

2. **Delete the IP tunnel.**

   ```
   # dladm delete-iptun tunnel-link
   ```

The only option for this command is `-t`, which causes the tunnel to be deleted temporarily. When you reboot the system, the tunnel is restored.

Example 39 Deleting an IPv6 Tunnel That is Configured With an IPv6 Interface

In the following example, a persistent tunnel is permanently deleted.

```
# ipadm delete-ip ip6.tun0
# dladm delete-iptun ip6.tun0
```

Index