Administering TCP/IP Networks, IPMP, and IP Tunnels in Oracle® Solaris 11.4

Part No: E60991
November 2020

ORACLE

Administering TCP/IP Networks, IPMP, and IP Tunnels in Oracle Solaris 11.4

Part No: E60991

Copyright © 2011, 2020, Oracle and/or its affiliates.

License Restrictions Warranty/Consequential Damages Disclaimer

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

Warranty Disclaimer

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

Restricted Rights Notice

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

Hazardous Applications Notice

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

Third-Party Content, Products, and Services Disclaimer

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Pre-General Availability Draft Label and Publication Date

Pre-General Availability: 2020-01-15

Pre-General Availability Draft Documentation Notice

If this document is in public or private pre-General Availability status:

This documentation is in pre-General Availability status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

Oracle Confidential Label

ORACLE CONFIDENTIAL. For authorized use only. Do not distribute to third parties.

Revenue Recognition Notice

If this document is in private pre-General Availability status:

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your pre-General Availability trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Master Agreement, Oracle License and Services Agreement, Oracle PartnerNetwork Agreement, Oracle distribution agreement, or other license agreement which has been executed by you and Oracle and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Référence: E60991

Copyright © 2011, 2020, Oracle et/ou ses affiliés.

Restrictions de licence/Avis d'exclusion de responsabilité en cas de dommage indirect et/ou consécutif

Ce logiciel et la documentation qui l'accompagne sont protégés par les lois sur la propriété intellectuelle. Ils sont concédés sous licence et soumis à des restrictions d'utilisation et de divulgation. Sauf stipulation expresse de votre contrat de licence ou de la loi, vous ne pouvez pas copier, reproduire, traduire, diffuser, modifier, accorder de licence, transmettre, distribuer, exposer, exécuter, publier ou afficher le logiciel, même partiellement, sous quelque forme et par quelque procédé que ce soit. Par ailleurs, il est interdit de procéder à toute ingénierie inverse du logiciel, de le désassembler ou de le décompiler, excepté à des fins d'interopérabilité avec des logiciels tiers ou tel que prescrit par la loi.

Exonération de garantie

Les informations fournies dans ce document sont susceptibles de modification sans préavis. Par ailleurs, Oracle Corporation ne garantit pas qu'elles soient exemptes d'erreurs et vous invite, le cas échéant, à lui en faire part par écrit.

Avis sur la limitation des droits

Si ce logiciel, ou la documentation qui l'accompagne, est livré sous licence au Gouvernement des Etats-Unis, ou à quiconque qui aurait souscrit la licence de ce logiciel pour le compte du Gouvernement des Etats-Unis, la notice suivante s'applique :

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

Avis sur les applications dangereuses

Ce logiciel ou matériel a été développé pour un usage général dans le cadre d'applications de gestion des informations. Ce logiciel ou matériel n'est pas conçu ni n'est destiné à être utilisé dans des applications à risque, notamment dans des applications pouvant causer un risque de dommages corporels. Si vous utilisez ce logiciel ou matériel dans le cadre d'applications dangereuses, il est de votre responsabilité de prendre toutes les mesures de secours, de sauvegarde, de redondance et autres mesures nécessaires à son utilisation dans des conditions optimales de sécurité. Oracle Corporation et ses affiliés déclinent toute responsabilité quant aux dommages causés par l'utilisation de ce logiciel ou matériel pour des applications dangereuses.

Marques

Oracle et Java sont des marques déposées d'Oracle Corporation et/ou de ses affiliés. Tout autre nom mentionné peut correspondre à des marques appartenant à d'autres propriétaires qu'Oracle.

Intel et Intel Inside sont des marques ou des marques déposées d'Intel Corporation. Toutes les marques SPARC sont utilisées sous licence et sont des marques ou des marques déposées de SPARC International, Inc. AMD, Epyc, et le logo AMD sont des marques ou des marques déposées d'Advanced Micro Devices. UNIX est une marque déposée de The Open Group.

Avis d'exclusion de responsabilité concernant les services, produits et contenu tiers

Ce logiciel ou matériel et la documentation qui l'accompagne peuvent fournir des informations ou des liens donnant accès à des contenus, des produits et des services émanant de tiers. Oracle Corporation et ses affiliés déclinent toute responsabilité ou garantie expresse quant aux contenus, produits ou services émanant de tiers, sauf mention contraire stipulée dans un contrat entre vous et Oracle. En aucun cas, Oracle Corporation et ses affiliés ne sauraient être tenus pour responsables des pertes subies, des coûts occasionnés ou des dommages causés par l'accès à des contenus, produits ou services tiers, ou à leur utilisation, sauf mention contraire stipulée dans un contrat entre vous et Oracle.

Date de publication et mention de la version préliminaire de Disponibilité Générale ("Pre-GA")

Version préliminaire de Disponibilité Générale ("Pre-GA") : 15.01.2020

Avis sur la version préliminaire de Disponibilité Générale ("Pre-GA") de la documentation

Si ce document est fourni dans la Version préliminaire de Disponibilité Générale ("Pre-GA") à caractère public ou privé :

Cette documentation est fournie dans la Version préliminaire de Disponibilité Générale ("Pre-GA") et uniquement à des fins de démonstration et d'usage à titre préliminaire de la version finale. Celle-ci n'est pas toujours spécifique du matériel informatique sur lequel vous utilisez ce logiciel. Oracle Corporation et ses affiliés déclinent expressément toute responsabilité ou garantie expresse quant au contenu de cette documentation. Oracle Corporation et ses affiliés ne sauraient en aucun cas être tenus pour responsables des pertes subies, des coûts occasionnés ou des dommages causés par l'utilisation de cette documentation.

Mention sur les informations confidentielles Oracle

INFORMATIONS CONFIDENTIELLES ORACLE. Destinées uniquement à un usage autorisé. Ne pas distribuer à des tiers.

Avis sur la reconnaissance du revenu

Si ce document est fourni dans la Version préliminaire de Disponibilité Générale ("Pre-GA") à caractère privé :

Les informations contenues dans ce document sont fournies à titre informatif uniquement et doivent être prises en compte en votre qualité de membre du customer advisory board ou conformément à votre contrat d'essai de Version préliminaire de Disponibilité Générale ("Pre-GA") uniquement. Ce document ne constitue en aucun cas un engagement à fournir des composants, du code ou des fonctionnalités et ne doit pas être retenu comme base d'une quelconque décision d'achat. Le développement, la commercialisation et la mise à disposition des fonctions ou fonctionnalités décrites restent à la seule discrétion d'Oracle.

Ce document contient des informations qui sont la propriété exclusive d'Oracle, qu'il s'agisse de la version électronique ou imprimée. Votre accès à ce contenu confidentiel et son utilisation sont soumis aux termes de vos contrats, Contrat-Cadre Oracle (OMA), Contrat de Licence et de Services Oracle (OLSA), Contrat Réseau Partenaires Oracle (OPN), contrat de distribution Oracle ou de tout autre contrat de licence en vigueur que vous avez signé et que vous vous engagez à respecter. Ce document et son contenu ne peuvent en aucun cas être communiqués, copiés, reproduits ou distribués à une personne extérieure à Oracle sans le consentement écrit d'Oracle. Ce document ne fait pas partie de votre contrat de licence. Par ailleurs, il ne peut être intégré à aucun accord contractuel avec Oracle ou ses filiales ou ses affiliés.

Accessibilité de la documentation

Pour plus d'informations sur l'engagement d'Oracle pour l'accessibilité de la documentation, visitez le site Web Oracle Accessibility Program, à l'adresse : http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info ou le site http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs si vous êtes malentendant.

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=E37838-01.

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 network protocols on Oracle Solaris systems. It assumes that a TCP/IP network is configured and operational, either as an IPv4-only or a dual-stack IPv4/IPv6 network.

The chapter contains the following topics:

- "Using Rights Profiles to Perform Network Configuration"
- "Customizing Protocol Properties"
- "Administering Transport Layer Services"
- "Monitoring and Analyzing the Network"

Other references:

- *Planning for Network Deployment in Oracle Solaris 11.4*
- *Configuring and Managing Network Components in Oracle Solaris 11.4*
- *Troubleshooting Network Administration Issues in Oracle Solaris 11.4*

Note - To administer TCP/IP networks and issue commands described in this chapter, you must have the appropriate rights profile as explained in the following section "Using Rights Profiles to Perform Network Configuration".

Using Rights Profiles to Perform Network Configuration

Oracle Solaris implements role-based access control (RBAC) to control system access. To perform tasks associated with network configuration, you must be assigned at least the Network Management profile. This profile is a superset that consists of other network-related profiles such as in the following partial list:

- Name Service Management for configuring name services.
- Network Wifi Management for configuring WiFi.
- Elastic Virtual Switch Administration for configuring the elastic virtual switch.

- Network Observability for accessing observability devices.

To obtain a complete list of the profiles in the Network Management profile, type:

```
$ profiles -p "Network Management" info
```

An administrator that has the `solaris.delegate.*` authorization can assign the Network Management profile to users to enable them to administer the network.

For example, an administrator assigns the Network Management rights profile to user `jdoe`. Before `jdoe` executes a privileged network configuration command, `jdoe` must be in a profile shell. The shell can be created by issuing the `pfbash` command. Or, `jdoe` can combine `pfexec` with every privileged command that is issued, such as `pfexec dladm`.

As an alternative, instead of assigning the Network Management profile directly to individual users, a system administrator can create a role that would contain a combination of required profiles to perform a range of tasks.

Suppose that a role `netadmin` is created with the profiles for network configuration as well as zone creation and configuration. User `jdoe` can issue the `su` command to assume that role. All roles automatically get `pfbash` as the default shell.

For more information about rights profiles, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.4*.

Customizing Protocol Properties

For most cases, the default values of protocol properties suffice to set up a functional network. However, these values can be reset and customized if necessary.

Commands for Setting Protocol Properties

To manage protocol properties, you use the following `ipadm` subcommands:

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

Displays current property settings of a protocol. If you omit the `-p` *property* option, then the property settings of all protocols are displayed.

`ipadm set-prop -p` *property=value*`[,` *property=value*`, ...]` *protocol*

Assigns a value to one or more properties of a protocol. To remove a property value, use the `-=` qualifier in the *property=value* pair.

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

Resets a specific protocol property to its default value.

See also *Configuring and Managing Network Components in Oracle Solaris 11.4*.

Configuring Protocol Properties: Sample Cases

The sample cases in this section describe ways to configure protocol properties.

Enabling Global Packet Forwarding

Packet forwarding is a property common to both the interface and the TCP/IP protocol so you can set the property's scope of implementation.

Packet forwarding on the interface level limits the function to that interface. If set on the protocol level, packet forwarding is global across all interfaces.

- On the interface level such as `net0`, you would use the `set-ifprop` subcommand:

```
$ ipadm set-ifprop -p forwarding=on -m ipv4|ipv6 net0
```

- On the protocol or global level, you would use the `set-prop` command:

```
$ ipadm set-prop -p forwarding=on ipv4|ipv6
```

The property can be set on both interface and protocol at the same time. Thus, although forwarding is enabled globally, you can still selectively implement the functionality on each interface.

Setting Up a Privileged Port

On transport protocols such as TCP, UDP, and SCTP, ports 1-1023 are privileged ports by default. Port numbers greater than 1023 are non-privileged.

You can extend the range of privileged ports beyond 1023, or mark specific ports in the non-privileged range as privileged. Processes that bind to a privileged port must be running with `root` permissions.

Setting up privileged ports involves the following properties:

`smallest-nonpriv-port`	Beginning of the range of non-privileged port numbers. By default, the port number is 1024.
`extra-priv-ports`	Ports outside of the privileged range that are set as privileged. You can assign multiple values to this property.

Suppose that you want to set TCP ports 3001 and 3050 as privileged ports, with access restricted to just the `root` role. First, you check the lowest number for a non-privileged port, which in the following output is 1024. Therefore, you can proceed with designating ports 3001 and 3050 as privileged.

```
$ 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 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   2049,4045    2049,4045   1-65535
                                  3001,3050   3001,3050
```

To remove an extra-privileged port, use the `-=` qualifier. For example:

```
$ 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   2049,4045    2049,4045   1-65535
                                  3001        3001
```

Implementing Traffic Congestion Control

Network congestion typically occurs in the form of router buffer overflows where nodes send more packets than the network can accommodate. Oracle Solaris supports the following algorithms that prevent traffic congestion by establishing controls on the sending systems:

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.

Algorithm	Oracle Solaris Name	Description
CUBIC	`cubic`	Currently the default algorithm in Linux 2.6. Changes the congestion avoidance phase from linear window increase to a cubic function.
Vegas	`vegas`	A classic delay-based algorithm that attempts to predict congestion without triggering actual packet loss.
DCTCP	`dctcp`	An algorithm that is based on the Explicit Congestion Notification (ECN) extension and is designed for traffic within a datacenter to predict and estimate congestion.

Beginning with Oracle Solaris 11.4, the Data Center TCP (DCTCP) algorithm is supported for TCP traffic *only*. To achieve the most benefits from using DCTCP, follow these guidelines:

- Use switches that support Explicit Congestion Notification (ECN) marking, as specified in Section 5 of RFC 3168 (`https://tools.ietf.org/html/rfc3168#section-5`).

  DCTCP requires that systems on both ends of the TCP connection have ECN. In Oracle Solaris, DCTCP auto-enables ECN for the specific connection that is using DCTCP. If the system on the other end is running a different OS, make sure ECN is enabled there also.
- DCTCP is beneficial where round-trip time (RTT) is low. Thus, lower configurations for the minimum Retransmission Timeout (min RTO) and delayed acknowledgement (ACK) timeout are recommended.

To apply congestion control, set the following TCP properties:

`cong-enabled` Comma-separated list of algorithms that are currently operational. Except for DCTCP, these algorithms apply to both TCP and SCTP traffic. You can specify multiple algorithms that you want to use.

`cong-default` Algorithm that is automatically used when applications do not explicitly any for socket options. The value applies to both global and non-global zones.

This property must always have a defined algorithm.

The following syntax adds (+=) or removes (-=) algorithms for congestion control.

```
$ ipadm set-prop -p cong-enabled+=|-=algorithm[,algorithm,...] tcp
```

To replace the default algorithm, type:

```
$ ipadm set-prop -p cong-default=algorithm tcp
```

Note - No sequence rules exist for add and removing algorithms.

In the following 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 cong_default,cong_enabled tcp
PROTO PROPERTY              PERM CURRENT      PERSISTENT   DEFAULT      POSSIBLE
tcp   cong-default           rw   newreno      newreno      newreno      newreno,cubic,
                                                                        dctcp,
                                                                        highspeed,
                                                                        vegas
tcp   cong-enabled           rw   newreno,     newreno,     newreno      newreno,cubic,
                                  cubic,dctcp, cubic,dctcp,              dctcp,
                                  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,
                                                                        dctcp,
                                                                        highspeed
tcp   cong-enabled           rw   newreno,     newreno,     newreno      newreno,cubic,
                                  cubic,dctcp, cubic,dctcp,              dctcp,
                                  highspeed    highspeed                 highspeed,
                                                                        vegas
```

Enabling Multiple Listeners on TCP, SCTP, and UDP Ports

In this release, the `SO_REUSEPORT` socket option can support two or more listeners to bind to the same address/port pair. For TCP and SCTP sockets, any new incoming connection requests are distributed across those listeners. For a UDP socket, any incoming datagrams are distributed across bound sockets.

Oracle Solaris supports the following `SO_REUSEPORT` algorithms:

- `rr` – Round robin
- `src-ip` – Hashing on source IP address
- `src-ip-port` – Hashing on source IP address and source port
- `src-dst-ip` – Hashing on source and destination IP address
- `src-dst-ip-ports` – Hashing on all four tuples

The following example shows how to change the algorithm that the `SO_REUSEPORT` option uses for UDP sockets. The current algorithm is first determined and then replaced.

```
$ ipadm show-prop -p reuseport-lbalg udp
PROTO PROPERTY              PERM CURRENT      PERSISTENT   DEFAULT      POSSIBLE
udp   reuseport-lbalg       rw   src-dst-     --           src-dst-     rr,src-ip,
                                  ip-ports                  ip-ports     src-ip-port,
                                                                         src-dst-ip,
                                                                         src-dst-ip-ports

$ ipadm set-prop -p reuseport-lbalg=src-dst-ip udp
$ ipadm show-prop -p reuseport-lbalg udp
PROTO PROPERTY              PERM CURRENT      PERSISTENT   DEFAULT      POSSIBLE
udp   reuseport-lbalg       rw   src-dst-ip   src-dst-ip   src-dst-     rr,src-ip,
                                                          ip-ports     src-ip-port,
                                                                       src-dst-ip,
                                                                       src-dst-ip-ports
```

Changing the TCP Receive Buffer Size

By default, the TCP receive buffer is 128 KB. Because applications do not use available bandwidth uniformly, a connection latency might require you to change the buffer size.

For example, Secure Shell performs additional checksum and encryption on the data stream, and consequently causes overhead on bandwidth use. Likewise, some applications perform bulk transfers. For both these cases, adjusting buffer sizes would increase efficiency in bandwidth use.

To calculate the appropriate receive buffer size, first use the `ping -s` *host* command to determine the value of the connection latency. Then, multiply the connection latency by the available bandwidth to obtain the bandwidth delay product (BDP). The appropriate receive buffer size approximates the BDP.

Note, however, that the use of bandwidth also depends on a variety of conditions which can further change the calculations.

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
```

Oracle Solaris does not provide a preferred set value for the buffer size 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.

Administering Transport Layer Services

The `inetd` daemon is responsible for starting standard Internet services when a system boots. These services include applications that use standard transport layer protocols. See the `inetd`(8) man page.

Recommendations for Systems That Run `inetd` Based Services

To safeguard against potential security vulnerabilities, limit the number of concurrent processes that are running necessary `inetd` based services. Also, disable any `inetd` based service that is not required.

The `inetadm` command configures all `inetd` based services. For reference, see the `inetadm`(8) man page.

`inetadm -p` lists the default settings of properties that are common to all `inetd` based services. To configure these properties, use the `inetadm -m` syntax.

```
$ 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
```

`max_copies` controls the number of processes that can run concurrently. The value `-1` indicates that the number is unlimited. To set limits to the `finger` service, for example, you would type the following:

```
$ 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

$ inetadm -m svc:/network/finger:default max_copies=3
$ inetadm -l finger | grep copies
max_copies=3
```

Note - Because the appropriate limit varies for customers and environments, no default value is provided.

Configuring `inetd` Based Services

The section "Modifying Services that are Controlled by inetd" in *Managing System Services in Oracle Solaris 11.4* includes steps for adding a new instance of an `inetd` based service.

The following procedure applies those steps to add an `echo` service to the SCTP protocol. This procedure can serve as a guide if you want to add other `inetd` services to transport layer protocols.

How to Add a New Instance of the `echo` Service to SCTP

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

1. (Optional) Confirm that the `echo` service is controlled by the `inetd` daemon.

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

2. Determine the location of the manifest for the `echo` service.

```
$ 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     <- location

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     <- location
```

3. Edit the `/lib/svc/manifest/echo.xml` file to add a new `echo` instance.

a. Copy and paste an existing `echo` element in the XML file.

`echo.xml` contains two `echo`-related elements: `dgram` and `stream`. You can use either to create a copy of the `echo` element on the file.

b. Give the new `echo` instance a unique name.

For this example, the name is `sctp_stream`.

c. Change the property values in `sctp_stream` as necessary.

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

The revisions in the `/lib/svc/manifest/echo.xml` file are shown in bold:

```
<instance name='sctp_stream' enabled='false' >
    <exec_method
 type='method'
 name='inetd_start'
 exec='/usr/lib/inet/in.echod -d'
 timeout_seconds='0'>
     <method_context>
  <method_credential user='root' group='root' />
     </method_context>
    </exec_method>
.
    <property_group name='inetd' type='framework'>
 <propval name='endpoint_type' type='astring' value='dgram' />
 <propval name='proto' type='astring' value='sctp' />
 <propval name='wait' type='boolean' value='true' />
    </property_group>
</instance>
```

4. Add the new `sctp_stream` to the `/etc/svc/profile/generic.xml` file.

The addition is shown in bold:

```
<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>
```

5. Restart the manifest import service.

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

6. Verify that the `sctp_stream` has been added.

```
$ svcs echo
```

7. Verify the property values of `sctp_stream`.

```
$ 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
```

8. **Add a definition for `sctp_stream` to the `/etc/services` file.**

   Use the format *service-name port/protocol* [*aliases*]. For example:

   ```
   echo    30/sctp
   ```

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

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

Logging IP Addresses of All Incoming TCP Connections

IP addresses for all incoming TCP connections are logged by the `rsyslog` service at the `daemon.notice` facility and level. The information is located in `/var/adm/messages`. See the `syslog.conf`(5) man page.

To trace all TCP connections, issue the following command:

```
$ inetadm -M tcp_trace=TRUE
```

Using the TCP ECN Feature

In Oracle Solaris 11.4, TCP actively includes Explicit Congestion Notification (ECN) negotiation with a peer when initiating a connection. ECN negotiation enables TCP to more accurately detect congestion on the network and react appropriately.

This feature requires that you use routers and switches that support ECN and allow ECN-enabled packets to pass through the network. Otherwise, remote connectivity might become problematic and you would need to adjust the ECN setting.

To detect whether a connection problem with a remote host is related to ECN, use the following approach:

1. Identify the target's IP address.

   For this example, assume from the host `source-server` you are connecting to `old-server` with the IP address 198.51.100.113.
2. Use the snoop command as follows:

```
$ snoop -V from 198.51.100.113 src tcp

Using device net3 (promiscuous mode)

________________________________
old-server.example.com -> source-server ETHER Type=0800 (IP), size=60 bytes
old-server.example.com -> source-server IP  D=198.51.100.124 S=198.51.100.113 LEN=40,
 ID=30421, TOS=0x0, TTL=25
old-server.example.com -> source-server TCP D=60726 S=2007 Rst Ack=3966642163 Win=0

$ snoop -v -I net3 from 198.51.100.113 src tcp

     output elided
.
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 = 22213
```

```
IP:   Flags = 0x0
IP:         .0.. .... = may fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 25 seconds/hops
IP:   Protocol = 6 (TCP)
IP:   Header checksum = 6157
IP:   Source address = 198.51.100.113, old-server.example.com
IP:   Destination address = 198.51.100.159, source-server.example.com
IP:   No options
IP:
```

output elided

The output's IP Header indicates that ECN is not supported.

3. Change the default ECN property setting and then retest connections to the target.

```
root:~$ ipadm set-prop -p ecn=passive tcp
```

Using TCP Wrappers in Oracle Solaris

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 by allowing or denying the connection depending on the origin of the request. Use TCP wrappers to protect daemons such as Telnet and the File Transfer Protocol (FTP). For information about TCP wrapper support for `sendmail`, see "Support for TCP Wrappers in Version 8.12 of sendmail" in *Managing sendmail Services in Oracle Solaris 11.4*.

Note - You cannot use TCP wrappers to protect Secure Shell (SSH) on Oracle Solaris systems. For more information, see "Replacing TCP Wrappers With sshd_config Entries" in *Managing Secure Shell Access in Oracle Solaris 11.4*.

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

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

1. **Set the `tcp_wrappers` property to `TRUE`.**

```
$ inetadm -M tcp_wrappers=TRUE
```

2. **Configure the TCP wrappers access control policy.**
   See the `hosts_access`(3) man page located in the `/usr/sfw/man` directory.

▼ How to Protect the FTP Network Service With TCP Wrappers

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

1. **Follow the instructions in the `/usr/share/doc/proftpd/modules/mod_wrap.html` module.**
   Since the module is dynamic, you must load it to use TCP wrappers with FTP.

2. **Load the module by adding the following instructions to the `proftpd.conf` file:**

```
<IfModule mod_dso.c>
    LoadModule mod_wrap.c
</IfModule>
```

3. **Restart the FTP service.**

```
$ svcadm restart svc:/network/ftp
```

Using the Network Data Path Bypass Capability for UDP

Oracle Solaris includes kernel network data path bypass capability for UDP. This capability is provided by the `net_kernel_bypass` library. Whenever an application that can use the bypass feature is started, the library is preloaded to ensure a successful data communication. Applications that use the network data path bypass capability do not require recompilation or any further changes.

A UDP socket that cannot use the bypass capability is treated as a normal socket. All future operations on that socket follow the normal kernel network socket path.

If the data path bypass capability is used, send and receive operations are performed only in the user context. Except for send and receive operations, all socket operations follow the normal kernel socket path.

Note - UDP applications must have the `PRIV_SYS_NET_CONFIG` privilege to use the bypass capability.

The bypass capability has the following limitations:

- Data path bypass requires specifying a network interface. Thus, the feature cannot be used by socket bind operations to wild card addresses.
- Packet fragmentation is currently unsupported.
- Data path bypass cannot be used on virtual interfaces, including IPMP interfaces and link aggregations. Dynamic reconfiguration of the interface card is also unsupported. It can be used only in the global zone.
- If the bypass feature is used together with other kernel features such as IPsec, Trusted Extension, and packet filtering, create the UDP bypass socket first before enabling those other features.
- The bypass socket feature does not generate error notifications, which can result in a missed ICMP error message, for example, "Destination unreachable".
- Packet sniffing through tools such as the `snoop` command is not supported.
- The bypass socket does not interact with the kernel flow created by the `flowadm` command. Thus, the `flowstat` command will not include statistics for the bypass socket.

To check UDP sockets that use this capability, see “Displaying Information About UDP Sockets That Use Network Data Path Bypass” on page 33.

Caution - System processes and applications that use any of the unsupported features that are listed should not use the `net_kernel_bypass` library.

For reference, see the `net_kernel_bypass(3LIB)` 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. The utility enables you to perform the following tasks:

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

The utility's `-M` option enables you to specify per socket Service Level Agreement (SLA) properties. When you use the option while setting other properties, a MAC flow for the socket is created. For example:

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

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:

```
$ pkg install pkg:/network/netcat
```

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

Monitoring and Analyzing the Network

This section describes selected tools you can use to monitor your network's components and its hosted traffic to gauges performance and detect potential problem areas.

Using `tshark` and Wireshark

`tshark` is a command-line network traffic analyzer that can capture packet data from a live network. A decoded form of the data is either printed to standard output or written to a file. In addition, `tshark` is capable of detecting, reading, and writing the same capture files as those that are supported by Wireshark.

Used without any options, `tshark` works similarly to the `tcpdump` command and also uses the same live capture file format, `libpcap`.

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, Wireshark displays packet data on a live network or from a previously saved capture file. Like the `tcpdump` utility and other tools, Wireshark uses the `libpcap` format for file captures. However, Wireshark is also capable of reading and importing several other file formats.

Both `tshark` and Wireshark provide several unique features, such as:

- Capability to assemble all of the packets in a TCP conversation and displaying the data in that conversation in ASCII, EBCDIC or hex format
- More fields that can be filtered than in other network protocol analyzers
- Richer syntax than other network protocol analyzers for creating filters

To use these tools, make sure that the `tshark` and `wireshark` software packages are installed.

For reference, see the Wireshark documentation at `http://www.wireshark.org/` and the `tshark`(1) and `wireshark`(1) man pages.

Using the `netstat` Command

The `netstat` command displays network status and protocol statistics of TCP, SCTP, and UDP endpoints as well as routing table and interface information.

By itself, the command displays the status of connected sockets on the system. To display the status of both connected and unconnected sockets, use the `netstat -a` syntax.

The command has other options which can be used alone or in combination to tailor the output to your preference. The examples that follow show how to use different `netstat` options.

For reference, see the `netstat`(8) man page.

Displaying the Status of Sockets

By default, the `netstat` command displays both IPv4 and IPv6 information. To limit the information to a specific IP version, do one of the following:

- Set the `DEFAULT_IP` variable in the `/etc/default/inet_type` file:

  `DEFAULT_IP=`*ip-version*

  ip-version can be `IP_VERSION4` or `IP_VERSION6`.

  For more information, see the `inet_type`(5) man page.
- Use the `-f` *version* option with the command. *version* can be one of the following:

  `inet` for IPv4
  `inet6` for IPv6
  `sdp` for Socket Description Protocol
  `unix` for UNIX domain sockets used for internal communications

EXAMPLE 1 Displaying Connected Sockets

This example shows how to limit the output to the status of connected 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
```

EXAMPLE 2 Displaying State Only of Sockets That Use the SO_REUSEPORT Mechanism

This example shows how to use the -L option to display information about sockets that are using the SO_REUSEPORT socket option. This option is currently only supported for TCP, UDP, and SCTP.

The -u option provides additional information about users, process IDs, and programs that either created the network endpoint or currently controls the network endpoint.

```
$ netstat -Lu
TCP: IPv4
Local    Remote
Address  Address  User     Pid     Command     Swind  Send-Q  Rwind  Recv-Q  State
-------  -------  -------  ------  ----------  -----  ------  ------ ------  ------
*.8001   *.*      userfoo  102185  web_server  0      0       128000 0       LISTEN
*.8001   *.*      userfoo  102188  web_server  0      0       128000 0       LISTEN
*.1234   *.*      userfoo  102138  foo_server  0      0       128000 0       LISTEN
*.1234   *.*      userfoo  101945  foo_server  0      0       128000 0       LISTEN
```

The output shows two TCP listeners on port 8001 and two TCP listeners on port 1234. Both ports are using the SO_REUSEPORT load balancing feature.

Displaying Statistics by Protocol

The -P *argument* option filters the output of netstat by protocol. *argument* can be the following:

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

For example, to display UDP output only for connected and unconnected sockets, you would type:

```
$ netstat -aP udp
UDP: IPv4

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

The output also includes statistics for send and receive buffers as well as information about transmit and receive overflows.

The counter for transmit overflows increases whenever IP cannot send the outgoing packet to the MAC layer due to unavailable space.

The counter for receive overflows increases whenever IP cannot send the incoming packet to the socket due to unavailable space. In such cases, the incoming packet is dropped.

Displaying Network Interface Status

The `-i` option displays the state of the network interfaces on the system and the number of packets passing through the interfaces.

This example displays the output pertaining only to the IPv4 traffic of `net0`:

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

The input packet count (`Ipkts`) can increase each time a client tries to boot. A count increase of input packets but a steady count of output packets (`Opkts`) indicate that the system does not know how to respond to request packets. This situation might be due to an incorrect address in the `hosts` or `ethers` database.

If the input packet count is steady over time, then the system does not see the packets at all and might indicate some type of failure, including hardware.

Displaying Information About UDP Sockets That Use Network Data Path Bypass

The `-k` option displays information about specific UDP sockets that use the network data path bypass capability. Without this option, these sockets are displayed like any other sockets.

For this feature's description, see “Using the Network Data Path Bypass Capability for UDP” on page 27.

```
$ netstat -aukP

UDP: IPv4
   Local Address  Remote Address  State  If  Ipkts   Opkts   Dpkts  User  Pid  Command
---------------   ------------   ------ ----- -----  ------  -----  ----- ---- --------
   *.64768                      Idle  net4   0     12556800  0    root 111530 udp_cli
```

Displaying the Status of Known Routes

The `-r` option displays the routing table, which lists all network routes that are known to a system.

```
$ netstat -r
Routing Table: IPv4
Destination           Gateway           Flags  Ref   Use   Interface
-------------------- -------------------- ----- ----- ------ ---------
host15               myhost               U         1  31059  net0
203.0.113.14         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 by the `netstat -r` command.

Parameter	Description
`Destination` `Destination/Mask`	Specifies the system 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.

Using the `ping` Command

The `ping` command helps to determine whether your system can exchange IP packets with a remote host. Through the command, the ICMP protocol sends a datagram to the target host and waits for a response. A response indicates available connectivity with that host.

When you issue the command, attempts to probe the remote host continue automatically for 20 seconds before the operation times out. You can specify a longer timeout period, which must be in seconds.

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

Sending an interrupt character immediately stops the operation.

For reference, see the `ping`(8) man page.

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

system. If the name server returns an IPv6 address for the target system, the command uses the IPv6 protocol. If the server returns only an IPv4 address, the command uses the IPv4 protocol.

To use a specific protocol, use the `-A` option and specify either `inet` or `inet6` for IPv4 and IPv6 protocols, respectively.

```
$ ping host -A inet|inet6
```

Investigating Dropped Packets

Packet loss can degrade network performance because additional time is spent retransmitting dropped data. The `-s` option reports packet loss between hosts.

```
$ ping -s host1.domain8
PING host1.domain8 : 56 data bytes
64 bytes from host1.example.COM (198.51.100.64): icmp_seq=0. time=1.67 ms
64 bytes from host1.example.COM (198.51.100.64): icmp_seq=1. time=1.02 ms
64 bytes from host1.example.COM (198.51.100.64): icmp_seq=2. time=0.986 ms
64 bytes from host1.example.COM (198.51.100.64): icmp_seq=3. time=0.921 ms
64 bytes from host1.example.COM (198.51.100.64): icmp_seq=4. time=1.16 ms
64 bytes from host1.example.COM (198.51.100.64): icmp_seq=5. time=1.00 ms
64 bytes from host1.example.COM (198.51.100.64): 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
```

By adding the `-W` option, you can specify a period in seconds in which the command waits for a response from the target host. By default, the wait period is 10 seconds.

Alternatively, you can use the `-w` option instead, where you specify a deadline, in seconds, before the ping operation ends, regardless of the number of packets sent or received.

Note - The `-W` or `-w` options must be used with either the `-s` or `-I` option.

If you use both `-W` or `-w` together in a single command, the `-W` option is ignored.

In the following example, the system waits 5 additional seconds for a response from the target after receiving no reply at the end of 10 ping requests:

```
$ ping -W 5 -s target 10
```

In the following example, the `ping` operation ends if one of the following conditions is met:

- The 10 second deadline elapses.
- The system receives 5 ping responses from the target
- System receives an error notification from the network

```
$ ping -w 10 -s target 5
```

Using the `traceroute` Command

The `traceroute` command traces the route that an IP packet follows to a remote system. Use this command to uncover any routing misconfiguration and routing path failures that make a remote host unreachable.

The 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 reference, see the `traceroute`(8) man page.

Command Modifications for IPv6 Support

You can use the `traceroute` command to trace both IPv4 and IPv6 routes to a specific system. Just like with the `ping` command, protocol selection depends on the addresses that are returned by the name server for the target system. If the name server returns an IPv6 address for the target system, the command uses the IPv6 protocol. If the server returns only an IPv4 address, the command uses the IPv4 protocol.

To use a specific protocol, use the `-A` option and specify either `inet` or `inet6` for IPv4 and IPv6 protocols, respectively.

```
$ traceroute destination-hostname -A inet|inet6
```

Discovering the Route to a Remote Host

The following sample output shows the seven-hop path that a packet follows from the local system to a remote system called `farhost`.

```
$ traceroute farhost
traceroute to farhost (198.51.100.39/27), 30 hops max, 40 byte packets
1  frbldg7c-86 (198.51.100.1/27)  1.516 ms  1.283 ms  1.362 ms
```

```
2  bldg1a-001 (198.51.100.2/27)  2.277 ms  1.773 ms  2.186 ms
3  bldg4-bldg1 (198.51.100.66/27)  1.978 ms  1.986 ms  13.996 ms
4  bldg6-bldg4 (198.51.100.132/27)  2.655 ms  3.042 ms  2.344 ms
5  farhost (198.51.100.39/27)  3.430 ms  3.312 ms  3.451 ms
```

The output also shows the time that it takes for a packet to traverse each hop.

Tracing All Routes

To trace all routes, use the -a option. The following example displays all of the possible routes to a dual-stack host called `v6host`:

```
$ traceroute -a v6host
traceroute: Warning: Multiple interfaces found; using 2001:db8:4a3a:1:56:a0:a8 @ net0:2
traceroute to v6rout86 (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..0.2.75),30 hops max,40 byte packets
1  v6-rout86 (198.51.100.1/27)  4.360 ms  3.452 ms  3.479 ms
2  flrmpj17u (198.51.100.131/27)  4.062 ms  3.848 ms  3.505 ms
3  farhost (203.0.113.23)  4.773 ms *  4.294 ms
4  distant (192..0.2.104)  5.128 ms  5.362 ms *
5  v6host  (192..0.2.85)  7.298 ms  5.444 ms *
```

Using the My Traceroute Utility

The My Traceroute (`mtr`) utility 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 and also tracks network hops between the current system and a target system. On the screen, the utility displays timing information which is updated constantly as new packets are sent out and responses are returned.

To use the `mtr` utility on your Oracle Solaris 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 reference, see the `mtr`(8) man page.

Using the `snoop` Command

The `snoop` command enables you to monitor network traffic by capturing network packets which are displayed or saved to a file.

In summary form, the displayed data pertains only 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 unless the verbose options are used.

When you issue the command, output is continuously generated until you send an interrupt character.

For reference, see the `snoop`(8) man page.

Command Modifications for IPv6 Support

By default, the `snoop` command displays both IPv4 and IPv6 packets. However, you can filter the displayed information to just IPv4 or IPv6 packets by specifying `ip` or `ip6`, respectively, with the command.

```
$ 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
```

Displaying Packets From All Interfaces

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

```
$ snoop
Using device /dev/net (promiscuous mode)
router5.local.com -> router5.local.com ARP R 203.0.113.13, 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..0.2.10.in-addr.arpa. Internet PTR ?
DNSserver.local.com  foohost       DNS R 192..0.2.10.in-addr.arpa. Internet PTR
niserve2.
.
```

```
.
fe80::a00:20ff:febb:e09 -> ff02::9 RIPng R (5 destinations)
```

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

Monitoring Packets on IP Layer Devices

Use the `-I` option to check network traffic on IP layer devices. These devices provide access to all of the packets with addresses that are associated with the network interface. The addresses include both IPv4 and IPv6 addresses. The addresses can be local or hosted on non-loopback interfaces or logical interfaces. The traffic can be loopback IP traffic, packets from remote machines, packets that are being sent from the system, or all forwarded traffic. In summary, you can monitor all traffic that is destined for the system. Type the following:

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

The `-V` and `-v` options generate verbose output.

interface has a wider scope than just physical IP interfaces. For example, an IPMP group is configured on an IPMP interface. Thus, this option enables you to monitor traffic from an IPMP group. You no longer need to monitor traffic separately on each underlying interface. Instead, the output is consolidated into a single output stream from the IPMP interface. For example:

```
$ snoop -I ipmp0 [-V|-v]
```

The following examples show different ways of monitoring network traffic on the IP layer.

EXAMPLE 3 Observing Traffic on the Loopback Interface

To obtain general and summary loopback traffic information, type the following:

```
$ 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
.
```

elided output

The output for IP layer information includes the `ipnet` header, which precedes the packets that are being observed. This header informs you of the source and destination of the traffic. In the sample output, the `0` ID of both `dli_srczone` and `dli_dstzone` indicates that the traffic is being generated from the global zone.

EXAMPLE 4 Observing Packet Flow for Interfaces in Local Zones

An administrator for the global zone can monitor traffic between zones, as well as within a zone. An administrator of a non-global zone can observe traffic that is sent and received by that zone.

This example shows zone traffic in the system. Both simple and verbose command syntax are used to display in different formats all packets that are associated with `net0`.

The example assumes the existence of two non-global zones: `sandbox` and `toybox`.

```
$ 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 = 198.51.100.1, 198.51.100.1
IP:   Destination address = 198.51.100.3, 198.51.100.3
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:
```

The `ipnet` header in the output indicates that the packet is coming from the global zone (ID `0` of `dli_srczone`) to `sandbox` (ID `1` of `dli_dstzone`).

Note - Using the `snoop` command with zone names is not supported.

Capturing `snoop` Output to a File

To direct `snoop` output to a file and analyze the file's contents, you use two command options separately. First you generate output to a file (`snoop -o`). Then you read the file's data (`snoop -i`).

```
$ snoop -o /tmp/cap
Using device /dev/eri (promiscuous mode)
30 snoop: 30 packets captured
.
.
^C
```

Note - 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.

Then you view the contents of the generated file.

```
$ snoop -i /tmp/cap
1   0.00000 fe80::a00:20ff:fee9:2d27 -> fe80::a00:20ff:fecd:4375
ICMPv6 Neighbor advertisement
...
10  0.91493    203.0.113.40 -> (broadcast)  ARP C Who is 203.0.113.40, 203.0.113.40 ?
34  0.43690 nearserver.example.com  -> 224.0.1.1  IP  D=224.0.1.1 S=203.0.113.40 LEN=28,
ID=47453, TO =0x0, TTL=1
35  0.00034  203.0.113.40 -> 224.0.1.1    IP  D=224.0.1.1 S=203.0.113.40 LEN=28,
 ID=57376,
TOS=0x0, TTL=47
```

Using the `ipstat` and `tcpstat` Commands

The `ipstat` and `tcpstat` commands monitor, gather, and report statistics about IP traffic. Each command provides different types of information.

- `ipstat` gathers and reports statistics about IP traffic, IP packet drops, and IP-related Management Information Base (MIB) events on a server. The reported information is based on the selected output mode and the sort order that you specify in the command syntax.

The command enables you to monitor network traffic and events at the IP layer. The data is aggregated on source, destination, higher-layer protocol, and interface. Use the `ipstat` command when you want to monitor the amount of traffic between servers.

- `tcpstat` gathers and reports statistics about TCP and UDP traffic, error events, and MIB events on a server based. The reported information is based on the selected output mode and the sort order that you specify in the command syntax. The command enables you to monitor network traffic and events at the transport layer, specifically for TCP and UDP. In addition to 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 two commands help you in the following ways:

- 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.
- Monitor packet drops and IP-related MIB events.

Note - The previous list is not exhaustive. You can use these commands in several other ways. For reference, see the `ipstat`(8) and `tcpstat`(8) man pages.

Examples in the Use of `ipstat`

The following examples show different ways to display IP statistics.

EXAMPLE 5 Displaying Statistics for All IP Communication

This example shows output of the `ipstat -c` command syntax. The option prints newer reports without overwriting previous reports. The number 3 indicates the interval in seconds for displaying data.

```
$ 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   203.0.113.255             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
```

EXAMPLE 6 Observing the Five Most Active IP Traffic Flows

The `-l` *n-lines* option specifies the top *n* most active IP traffic.

```
$ ipstat -l 5
SOURCE                    DEST                       PROTO    IFNAME     RATE
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 7 Displaying a Time Stamp

This example reports the top IP traffic with a time stamp in standard date format (`-d d`).

To print the timestamp in seconds or UNIX time, use `-d u`. The interval count is set to 10 seconds.

```
$ ipstat -d d -c 10
Monday, March 26, 2012 08:34:07 PM EDT
SOURCE                    DEST                       PROTO    IFNAME     RATE
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 8 Reporting IP Drops in Packets

This example shows how to use the `ipstat` command to display the number of packets that are dropped for each source, destination, and protocol combination.

```
$ ipstat -e
SOURCE                    DEST                   PROTO IFNAME    RATE   EVENT
-------------------------------------------------------------------------------
fe80::214:4fff:fe40:d0c8   ff02::202             IP    net0        6.0  drop-in
systl                     ntp.mcast.net          IGMP  net0        1.0  drop-in
tes-01-11.company.us.com   all-systems.mcast.net  IGMP  net0        1.0  drop-in
Total: bytes in:  9.0  bytes out:  0.0
```

Examples in the Use of `tcpstat`

The following examples show different ways to display TCP and UDP statistics.

EXAMPLE 9 Displaying Statistics for All TCP and UDP Communication

Compare the information here with the output in Example 5, “Displaying Statistics for All IP Communication,” on page 43 that also used the `-c` option.

```
$ tcpstat -c 3
ZONE         PID PROTO  SADDR             SPORT DADDR             DPORT   RATE
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
```

EXAMPLE 10 Observing the Five Most Active TCP Traffic Flows

This 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 11 Displaying Timestamp Information

In this example, the `tcpstat` command displays the 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   RATE
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
```

EXAMPLE 12 Reporting TCP and UDP Statistics by Packet Count

The -k option displays the number of packets instead of number of bytes that are exchanged.

```
$ tcpstat -k
ZONE         PID PROTO  SADDR             SPORT DADDR             DPORT   RATE
--------------------------------------------------------------------------------
global    100574 TCP    syst1                22 dhcp-system1     59198       2
global    100574 TCP    dhcp-system1      59198 systl              22       2
global    100531 UDP    10.5.238.52       46066 10.255.255.255     111       1
Total: packets in: 3 out: 2
```

EXAMPLE 13 Reporting Traffic for TCP-Related Events Grouped by Flow

This example shows how to display the rate at which transport layer events occur, grouped by flow. The flows are displayed with numeric source and destination IP addresses instead of their respective host names. To display only a subset of the events, instead of specifying `all`, provide a comma separated list of event names after the -E option. For a complete list of possible event names, use the `tcpstat -L` command.

```
$ tcpstat -E all -T tcp -n -g
ZONE       PID PROTO SADDR          SPORT  DADDR          DPORT RATE            EVENT
--------------------------------------------------------------------------------
global  100519 TCP   10.132.148.89  39443  10.134.71.92      22    5 tcpInInorderSegs
global  100519 TCP   10.132.148.89  39443  10.134.71.92      22    1     tcpInAckSegs
global  100519 TCP   10.132.148.89  39443  10.134.71.92      22    1     tcpRttUpdate
global  100519 TCP   10.134.71.92      22  10.132.148.8   39443    4        tcpOutAck
global  100519 TCP   10.134.71.92      22  10.132.148.89  39443    1   tcpOutDataSegs

Total packets: 12
```

Tracing and Logging IP Operations

This section describes additional methods for checking network operations.

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 `in.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 47 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   203.0.113.112    -->203.0.113.0/25
<UP|BROADCAST|RUNNING|MULTICAST|IPv4>
turn on RIP
Add    203.0.113.0        -->203.0.113.112      metric=0  net0  <NET_SYN>
Add    203.0.113.85/25  -->203.0.113.112      metric=0  net0  <IF|NOPROP>
```

Tracing 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.

Before you start the trace, disable the NDP service first. Then, after running the trace for a determined period, restart the service. Stop the trace anytime by sending an interrupt character. For example:

```
$ svcadm disable ndp
$ /usr/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

^C

$ svcadm enable ndp
```

♦ ♦ ♦ CHAPTER 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:

Note - To administer IPMP and issue commands described in this chapter, you must have the appropriate rights profile. See "Using Rights Profiles to Perform Network Configuration".

What's New in IPMP

In the current release, you can enable probe-based failure detection by individual IPMP groups.

A new interface property, `allow-xprobe`, has been added to Oracle Solaris. This enhancement enables you to allow or disallow transitive probing on a per-IPMP group basis. Previously, you could only enable transitive probing on a per-system or per-exclusive zone basis by setting the value in the `svc:/network/ipmp/config/transitive-probing` Service Management Facility (SMF) property. The previous method unnecessarily limited the flexibilities of IPMP configuration.

For more information, see "Selecting a Failure Detection Method" on page 84.

IPMP Support in Oracle Solaris

IPMP support in Oracle Solaris includes the following features:

- 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, the types of failure detection that are being used, and which interfaces have failed. See “Monitoring IPMP Information” on page 88.
- You can assign a custom name to an IPMP interface to identify the IPMP group more easily. See “Configuring IPMP Groups” on page 69.

Functions of an IPMP Configuration

IPMP configures together multiple IP interfaces into an *IPMP group*. The group functions like an IP interface, with data addresses to send or receive network traffic. The multiple underlying interfaces of the IPMP group ensures continuous network availability, If one underlying interface fails, the data addresses are redistributed amongst the remaining underlying active interfaces in the group. Thus, 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 for improving network performance and availability. For a comparison of these two technologies, see Appendix B, "Link Aggregations and IPMP: Feature Comparison," in *Managing Network Datalinks in Oracle Solaris 11.4*.

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 availability of the interface. See "Types of IPMP Interface Configurations" on page 53.

   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.

For example, use the following command to push the modules of each interface in the standard order for the IPMP group:

```
$ ifconfig net0 modinsert vpnmod@3
```

To plan an IPMP group, see "How to Plan an IPMP Group" on page 69.

IPMP Components

The IPMP software components are as follows:

- **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 has 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. See "Failure Detection in IPMP" on page 61.

  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 were 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 54 and the `in.mpathd(8)` man page.

- **IP kernel module** – Manages outbound load spreading by distributing the connection over the IPMP group interface across the set of available underlying IP interfaces within 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 behavior of the `mpathd` daemon.

  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 86.
- `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. See "Monitoring IPMP Information" on page 88 and the `ipmpstat(8)` 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 - Be 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 availability of the interface, 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 in 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 *Concepts for Oracle Solaris Cluster* in Oracle Help Center.

An IPMP group without underlying interfaces can also exist, such as a group whose underlying interfaces have been removed. The IPMP group is not destroyed, but the group cannot 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.

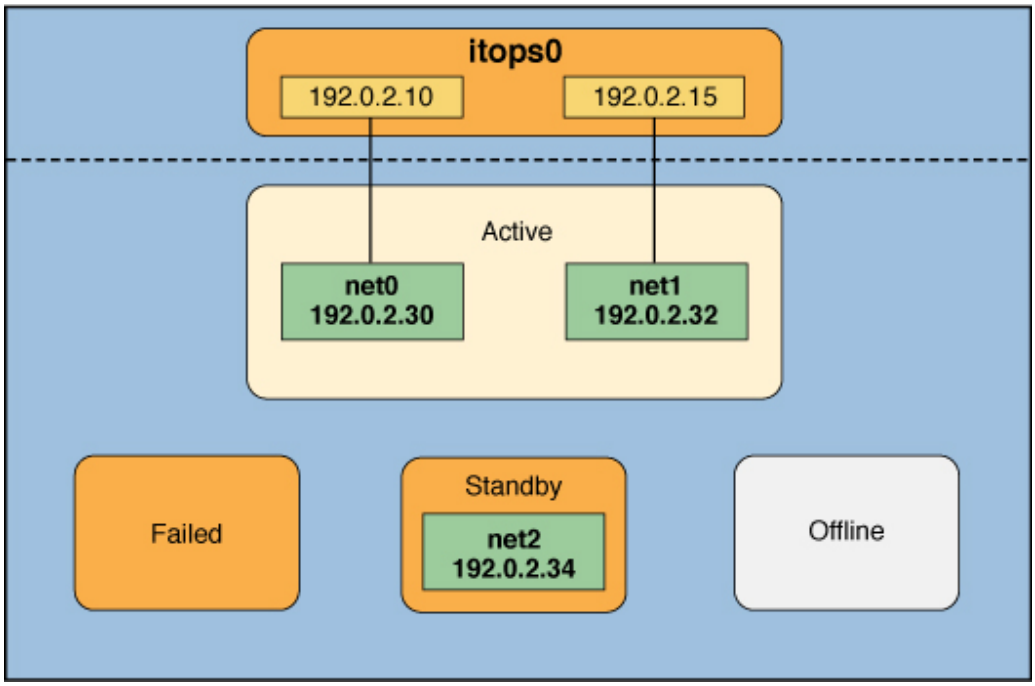
How IPMP Works

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 is 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: `192.0.2.10` and `192.0.2.15`.
- 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`: `192.0.2.30`
  - `net1`: `192.0.2.32`
  - `net2`: `192.0.2.34`

Note - The active, offline, standby, and failed areas in Figure 1, "IPMP Active-Standby Configuration," on page 55, Figure 2, "Interface Failure in IPMP," on page 57, Figure 3, "Standby Interface Failure in IPMP," on page 58, and Figure 4, "IPMP Recovery Process," on page 59 indicate only the status of underlying interfaces and not 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. See "Monitoring IPMP Information" on page 88.

For example, the following command displays information about the IPMP configuration, as shown in Figure 1, "IPMP Active-Standby Configuration," on page 55:

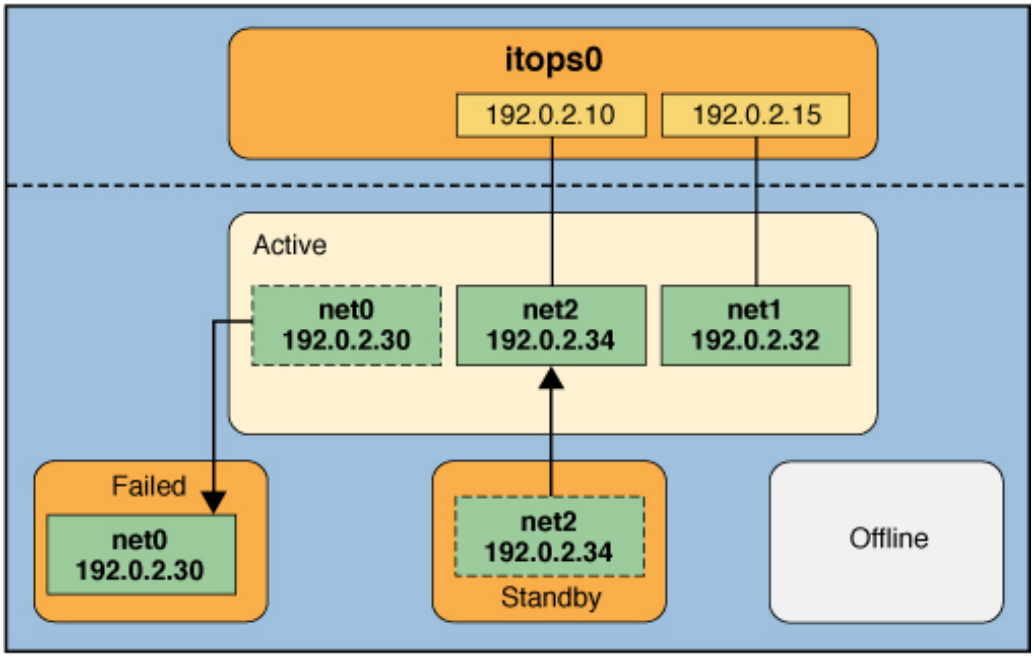
```
$ ipmpstat -g
GROUP     GROUPNAME     STATE     FDT        INTERFACES
itops0    itops0        ok        10.00s     net1 net0 (net2)
```

The following command displays the underlying interfaces in a group:

```
$ 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



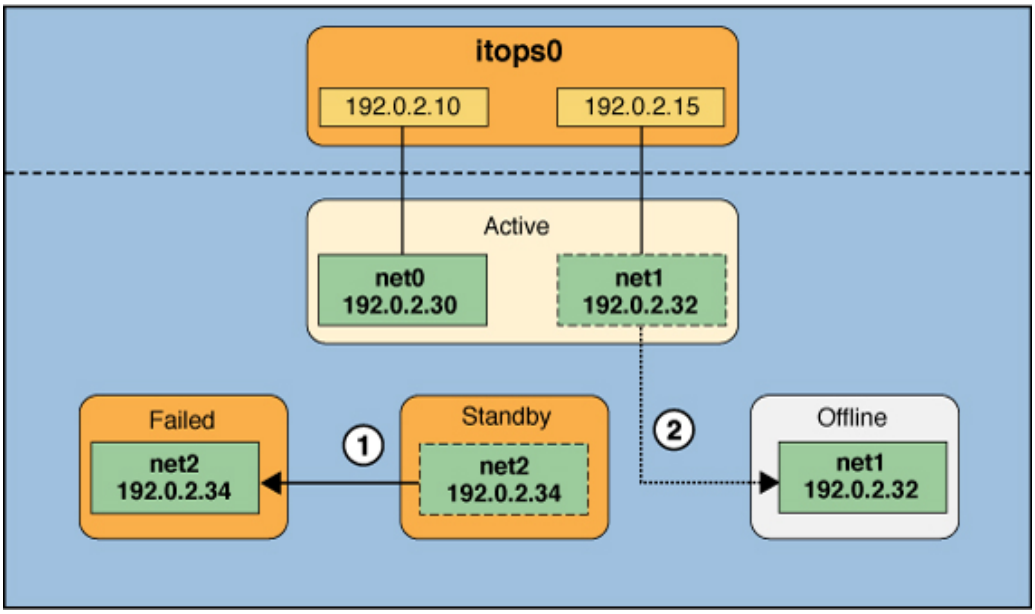
For the previous figure, the `ipmpstat` command displays the following information:

```
$ 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.

See Figure 3, “Standby Interface Failure in IPMP,” on page 58 for a different failure scenario, 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

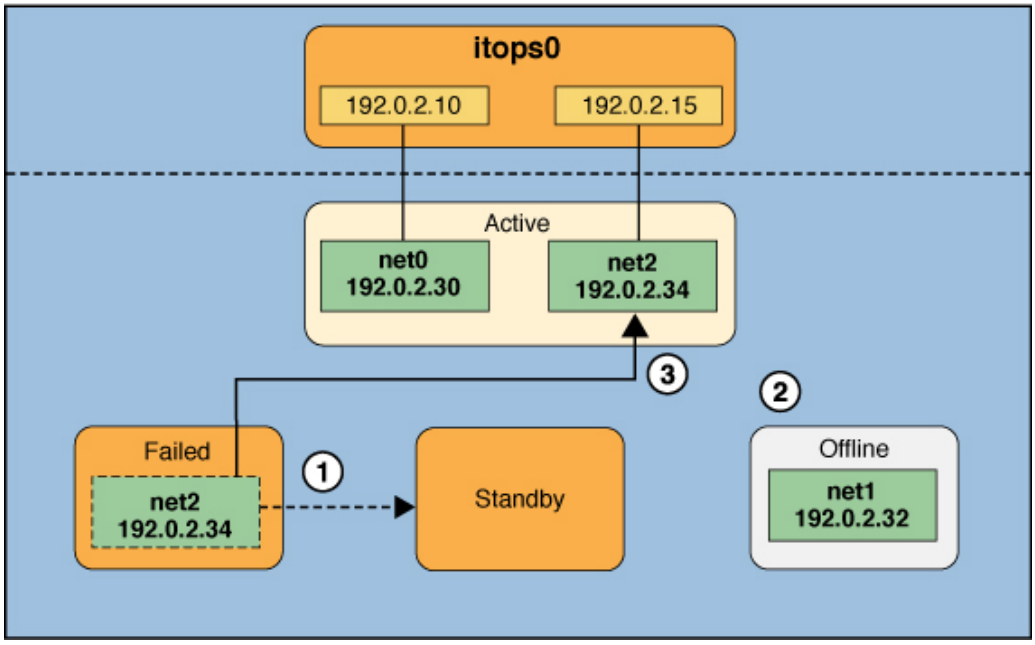


For the previous figure, the `ipmpstat` command displays the following information:

```
$ 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 the interface is repaired is different. The recovery process depends on the original number of active interfaces in the IPMP group compared with the configuration after the repair. The following figure represents the recovery process.

FIGURE 4 IPMP Recovery Process



In this recovery process, when `net2` is repaired, it normally reverts to its original status as a standby interface. However, the IPMP group still does not reflect the original number of two active interfaces because `net1` continues to remain offline. Thus, IPMP instead deploys `net2` as an active interface.

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

```
$ 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 57 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 following recovery information:

```
$ 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, see "FAILBACK=no Mode" on page 65.

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 both data addresses and test addresses. IP addresses reside on the IPMP interface group only and are specified as data addresses. Test addresses are IP addresses that reside on the underlying interfaces.

Note - Data and test addresses are not required to be DNS resolvable.

Data Addresses

IPMP *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 that you manually assign with the `ipadm` command. Data addresses are assigned to the IPMP interface group only. The standard IPv4 packet traffic and IPv6 packet traffic 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

IPMP *test addresses* are 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.

You must 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. See "Probe-Based Failure Detection" on page 62.

Previously for IPMP, test addresses had to be marked as `DEPRECATED` to avoid being used by applications, especially during interface failures. Now, 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, see 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 by using the `ipadm` command.

When an IPMP group has both IPv4 and IPv6 plumbed on all the 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 Oracle VM Server for SPARC software supports link-based IPMP with virtual network devices. You can configure the IPMP group to use link-based and probe-based failure detection

when you configure an IPMP group with virtual network devices. For more information, see *Oracle VM Server for SPARC 3.6 Administration Guide*.

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

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 or not test addresses are configured. Probe-based failure detection that does not use test addresses is referred to as *transitive probing*.

In Oracle Solaris, probe-based failure detection operates with test addresses by default. 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 84.

Probe-Based Failure Detection Using Test Addresses

This type of failure detection 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. See "How to Configure the Behavior of the IPMP Daemon" on page 86.

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 83.

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.

  For information about the packet format of transitive probes, see Appendix C, "Packet Format of Transitive Probes," in *Managing Network Datalinks in Oracle Solaris 11.4*.

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

IPMP also supports link-based failure detection and this type of failure detection is always enabled, provided that it is supported by the NIC driver.

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, the IPMP daemon can be configured 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 86.

Detecting Physical Interface Repairs

The *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 as follows, depending on how the failed interface was originally configured:

- 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. See "`FAILBACK=no` Mode" on page 65.

- 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 more information, see "How IPMP Works" on page 54.

`FAILBACK=no` Mode

By default, active interfaces that have failed and then been automatically repaired 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. See "How to Configure the Behavior of the IPMP Daemon" on page 86.

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. See the cfgadm(8) 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. You can configure and administer IPMP by using the `ipadm` command.

This chapter contains the following topics:

Note - To administer IPMP and issue commands described in this chapter, you must have the appropriate rights profile. See "Using Rights Profiles to Perform Network Configuration".

Configuring IPMP Groups

This section contains procedures for configuring IPMP groups.

▼ 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. See "Rules for Using IPMP" on page 51.

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

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 11.4*.

3. **Ensure that the same set of STREAMS modules is configured and pushed on all interfaces in the IPMP group.**
   For more information, see "Rules for Using IPMP" on page 51.

4. **Use the same IP addressing format on all the interfaces in the IPMP group.**
   If one interface is configured for IPv4, then you must configure all the interfaces in the IPMP group for IPv4. Likewise, if you add IPv6 addressing to one interface, then you must configure all 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 61.

6. **Ensure that all 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 53.

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. 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 53. The following procedure describes how to configure an active-standby IPMP group with DHCP.

Before You Begin Before configuring an IPMP group that uses DHCP, perform the following task:

- 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 11.4*. 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.
- 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 11.4*.
- 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 be used. If link-based failure detection discovers that the interface is functioning, the daemon might incorrectly report that the interface has been repaired. For more information about configuring DHCP, see *Working With DHCP in Oracle Solaris 11.4*.

Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

1. **Create an IPMP interface.**

```
$ ipadm create-ipmp ipmp-interface
```

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`.

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

```
$ ipadm create-ip under-interface
```

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

3. **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.

4. **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 11.4*.

5. **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 the previous step uses the format *under-interface*/*address-type*, for example, `net0/v4`.

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

Example 14 Configuring an IPMP Group With DHCP

This example shows the configuration of an active-standby IPMP group with DHCP.

- 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.

```
$ ipadm create-ipmp ipmp0

$ ipadm create-ip net0
$ ipadm create-ip net1
$ ipadm create-ip net2
$ ipadm add-ipmp -i net0 -i net1 -i net2 ipmp0

$ ipadm create-addr -T dhcp ipmp0
ipadm: ipmp0/v4
$ ipadm create-addr -T dhcp ipmp0
ipadm: ipmp0/v4a

$ 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

$ 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.

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 11.4*. 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 11.4*.

Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

1. **Create an IPMP interface.**

```
$ ipadm create-ipmp ipmp-interface
```

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`.

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

   `$ ipadm add-ipmp -i` *under-interface*`1 [-i` *underinterface*`2 ...]` *ipmp-interface*

   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.

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

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

   address can be in CIDR notation.

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

4. **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*

   address can be in CIDR notation. All test IP addresses in an IPMP group must belong to a single IP subnet and therefore use the 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 53.

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

1. **Create an IPMP interface.**

   $ **ipadm create-ipmp** *ipmp-interface*

   ipmp-interface specifies the name of the IPMP interface.

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

   $ **ipadm add-ipmp -i** *under-interface1* **[-i** *underinterface2* **...]** *ipmp-interface*

   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.

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

   $ **ipadm create-addr -a** *address ipmp-interface*

   address can be in CIDR notation.

4. **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*

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

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

   $ **ipadm set-ifprop -p standby=on -m ip** *under-interface*

Example 15 Configuring an Active-Standby IPMP Group

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

```
$ ipadm create-ipmp ipmp0

$ ipadm create-ip net0
$ ipadm create-ip net1
$ ipadm create-ip net2
$ ipadm add-ipmp -i net0 -i net1 -i net2 ipmp0

$ ipadm create-addr -a 192.0.2.5/24 ipmp0
```

```
ipadm: ipmp0/v4
$ ipadm create-addr -a 192.0.2.10/24 ipmp0
ipadm: ipmp0/v4a

$ ipadm create-addr -a 192.0.2.15/27 net0
ipadm: net0/v4
$ ipadm create-addr -a 192.0.2.20/27 net1
ipadm: net1/v4
$ ipadm create-addr -a 192.0.2.25/27 net2
ipadm: net2/v4

$ ipadm set-ifprop -p standby=on -m ip net2

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

$ ipmpstat -t
INTERFACE  MODE     TESTADDR        TARGETS
net0       routes   192.0.2.15/27   192.0.2.2/27
net1       routes   192.0.2.20/27   192.0.2.2/17
net2       routes   192.0.2.25/27   192.0.2.5/27
```

Maintaining IP Connectivity and Routing While Deploying IPMP

You can add an IP interface to an IPMP group by using 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        --      192.0.2.10/27

$ ipadm delete-addr net0/v4
$ ipadm create-addr -T static -a local=192.0.2.10/27 ipmp0/v4

$ ipadm
NAME      CLASS/TYPE  STATE    UNDER     ADDR
...
ipmp0     ipmp        ok       --        --
ipmp0/v4  static      ok       --        192.0.2.10/27
net0      ip          ok       ipmp0     --
```

Also, be mindful that any routes that you have 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.

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See “Using Rights Profiles to Perform Network Configuration” on page 13.

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 16 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        192.0.2.1/27     UG      107    176682262   net0
192.0.2.0/27   192.0.2.30/27    U        22    137738792   net0

$ route -p delete default 192.0.2.1/27 -ifp net0
$ route -p add default 192.0.2.1/27

$ netstat -nr
Routing Table: IPv4
Destination     Gateway       Flags    Ref     Use       Interface
-------------  ------------  --------  -----  -----------  --------
default        192.0.2.1/27     UG      107    176682262
192.0.2.0/27   192.0.2.30/27    U        22    137738792   net0
```

Administering IPMP

This section contains procedures for the administration of IPMP groups.

▼ How to Add an Interface to an IPMP Group

Before You Begin Check that the interface that you add to the group meets all of the necessary requirements. See "How to Plan an IPMP Group" on page 69.

Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

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

   $ **ipadm create-ip** *under-interface*

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

   $ **ipadm add-ipmp -i** *under-interface ipmp-interface*

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

Example 17 Adding an Interface to an IPMP Group

The following example shows how to 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
```

Removing an Interface From an IPMP Group

To remove one or more interfaces from the IPMP group, use the following syntax:

$ **ipadm remove-ipmp -i** *under-interface*[**-i** *under-interface* **...**] *ipmp-interface*

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 the 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.

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
```

Adding 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.

To add data addresses to the group, type:

```
$ 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.

To add test addresses to the group, type:

```
$ 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.

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

1. **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.

2. **Remove the IP addresses from an IPMP group.**

   - **Remove data addresses as follows:**

     ```
     $ ipadm delete-addr addrobj
     ```

     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
     ```

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

Example 18 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 15, “Configuring an Active-Standby IPMP Group,” on page 75. This example removes the test address from the underlying interface, `net1`.

```
$ ipadm show-addr net1
ADDROBJ           TYPE     STATE     ADDR
net1/v4           static   ok        192.0.2.25/27

$ ipadm delete-addr net1/v4
```

Moving an Interface Between IPMP Groups

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.

Use the following command syntax:

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

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.

In this 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 place `net0` in the IPMP group `cs-link1`.

```
$ ipadm add-ipmp -i net0 cs-link1
```

How to Delete an IPMP Group

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

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

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

   ```
   $ ipmpstat -g
   ```

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

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

   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.

3. **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 19 Deleting an IPMP Interface

This 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 describes probe-based failure detection in IPMP groups. It also provides procedures for setting failure detection for these groups.

About Probe-Based Failure Detection

Probe-based failure detection involves the use of target systems, as described in "Probe-Based Failure Detection" on page 62. In identifying targets for probe-based failure detection, the `in.mpathd` daemon operates in two modes: *router target mode* and *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 startup. 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 85 shows the exact syntax to use to add persistent routes to targets for probe-based failure detection. See "Maintaining IP Connectivity and Routing While Deploying IPMP" on page 76 and the `route`(8) man page.

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.
- Make sure that the netmask and broadcast addresses of the target systems are the same as the addresses in the IPMP group.
- Make sure the target system is 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, make sure that your probe targets meet the requirements that are listed in "Requirements for Choosing Targets for Probe-based Failure Detection" on page 84.

To use transitive probing, follow these steps:

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
```

See the in.mpathd(8) man page.

2. Remove any existing test addresses that have been configured for the IPMP group.

```
$ ipadm delete-addr address addrobj
```

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

To use test addresses to probe for failure, assign test addresses to the underlying interfaces of the IPMP group.

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

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 84.

Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

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 system that is to be used as a target in probe-based failure detection.**

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

destination-IP and *gateway-IP* are IPv4 addresses of the system to be used as a target. For example, you would type the following command to specify the target system `192.0.2.8/27`, which is on the same subnet as the interfaces in IPMP group, `ipmp0`:

```
$ route -p add -host 192.0.2.8/27 192.0.2.8/27 -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`

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

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

```
$ pfedit /etc/default/mpathd
```

See the `pfedit`(8) 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
  ```

  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` 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 65.

`no` — Indicates that data traffic does not return to a repaired interface. When a failed interface 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` — 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, see "Failure Detection and the Anonymous Group Feature" on page 64.

2. 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 you can 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. See the `ipmpstat(8)` 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 20, “Displaying IPMP Group Information,” on page 89.

`-a` Displays the data addresses that are configured for the IPMP groups. See Example 21, “Displaying IPMP Data Address Information,” on page 90.

`-i` Displays information about IP interfaces that are related to IPMP configuration. See Example 22, “Displaying Information About Underlying IP Interfaces of an IPMP Group,” on page 90.

`-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 23, “Obtaining IPMP Probe Target Information,” on page 92.

`-p` Displays information about the probes that are being used for failure detection. See Example 24, “Observing IPMP Probes,” on page 94.

EXAMPLE 20 Displaying 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. See the `in.mpathd`(8) 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 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 21 Displaying 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
192.0.2.5/27    up       ipmp0         net0    net0 net1
192.0.2.7/27    up       ipmp0         net1    net0 net1
192.0.2.35/27   up       acctg1        --          --
192.0.2.37/27   up       acctg1        --          --
192.0.2.15/27   up       field2        net2    net2 net7
192.0.2.20/27   up       field2        net7    net2 net7
192.0.2.25/27   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 22 Displaying 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 is in use (`yes`) or not (`no`).

`GROUP` Specifies the IPMP interface name. For anonymous groups, this field is empty. See the `in.mpathd`(8) man page.

`FLAGS` Indicates the status of each underlying interface, which can be one or any combination of the following flags:

- `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 can 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 23 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        192.0.2.15/27   192.0.2.1/27 192.0.2.3/27
net1        disabled      --              --
net3        disabled      --              --
net4        routes        192.0.2.35/27   192.0.2.37/27
```

```
net2        multicast     192.0.2.15/27   192.0.2.1 192.0.2.2
net6        multicast     192.0.2.20/27   192.0.2.2 192.0.2.1
net5        multicast     192.0.2.25/27   192.0.2.1 192.0.2.2
```

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       192.0.2.70/27    192.0.2.65/27
```

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 62.

`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 24 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 have appropriate privileges or use the `root` role to run this command.

The following example shows 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     192.0.2.1/27
0.17s   net4        612       --       --       --         192.0.2.1/27
0.25s   net2        602       0.61ms   1.10ms   1.10ms     192.0.2.3/27
0.26s   net6        602       --       --       --         192.0.2.3/27
0.25s   net5        601       0.62ms   1.20ms   1.00ms     192.0.2.1/27
0.26s   net7        603       0.79ms   1.11ms   1.10ms     192.0.2.1/27
1.66s   net4        613       --       --       --         192.0.2.3/27
1.70s   net0        603       0.63ms   1.10ms   1.10ms     192.0.2.35/27
^C
```

The following example shows 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     192.0.2.1/27
^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. You would run the `ipmpstat` command from a script. The script displays the failure detection time for an IPMP group, as shown in the following example:

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

♦ ♦ ♦ CHAPTER 4

About IP Tunnel Administration

This chapter provides an overview of IP tunnel administration in Oracle Solaris.

This chapter contains the following topics:

IP Tunnel Feature Summary

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:

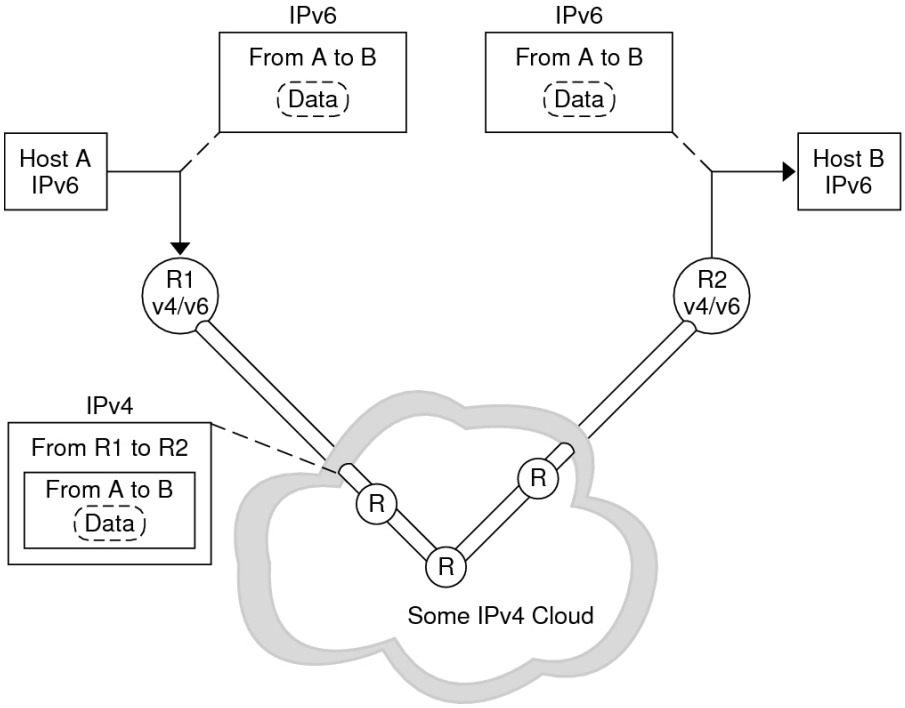
- **IPv4 tunnels** – IPv4 packets are encapsulated in an IPv4 header and sent to a preconfigured unicast IPv4 destination. To indicate more specifically the packets that flow over the tunnel, IPv4 tunnels are also called either *IPv4 over IPv4 tunnels* or *IPv6 over IPv4 tunnels*.
- **6to4 tunnels** – 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 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.

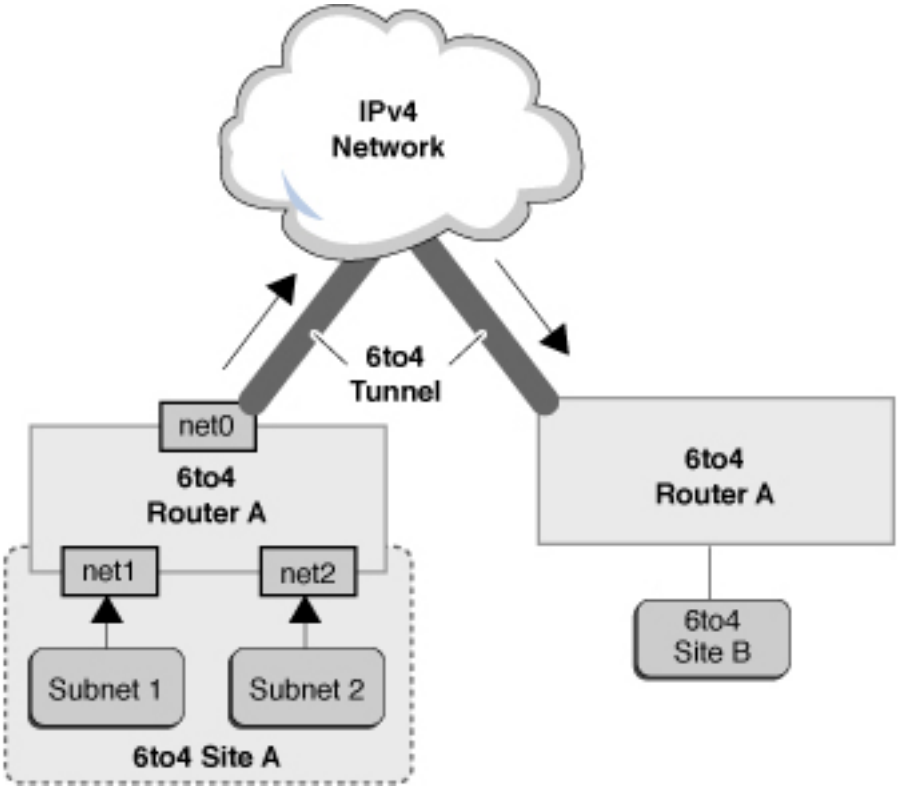
About 6to4 Tunnels

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. Thereafter, 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.

Before an IPv6 site can become 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, a boundary router on Site B must be configured for 6to4 support. Otherwise, packets that the router receives from Site A are not recognized and are then dropped.

About the `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 - Tunneling to a relay router is disabled by default in Oracle Solaris because the use of relay routers is insecure. 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 104. 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 110.

For more information, see the `6to4`(4M) man page.

Packet Flow Through the 6to4 Tunnel

This section describes the flow of packets from a system at one 6to4 site to a system at a remote 6to4 site. This scenario uses the topology that is shown in Figure 6, "Tunnel Between Two 6to4 Sites," on page 102. Moreover, the scenario assumes that the 6to4 routers and the 6to4 systems are already configured.

The packet flow is as follows:

1. A system on Subnet 1 of 6to4 Site A sends a transmission to a system at 6to4 Site B as the destination. Each packet header has a 6to4-derived source address and 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

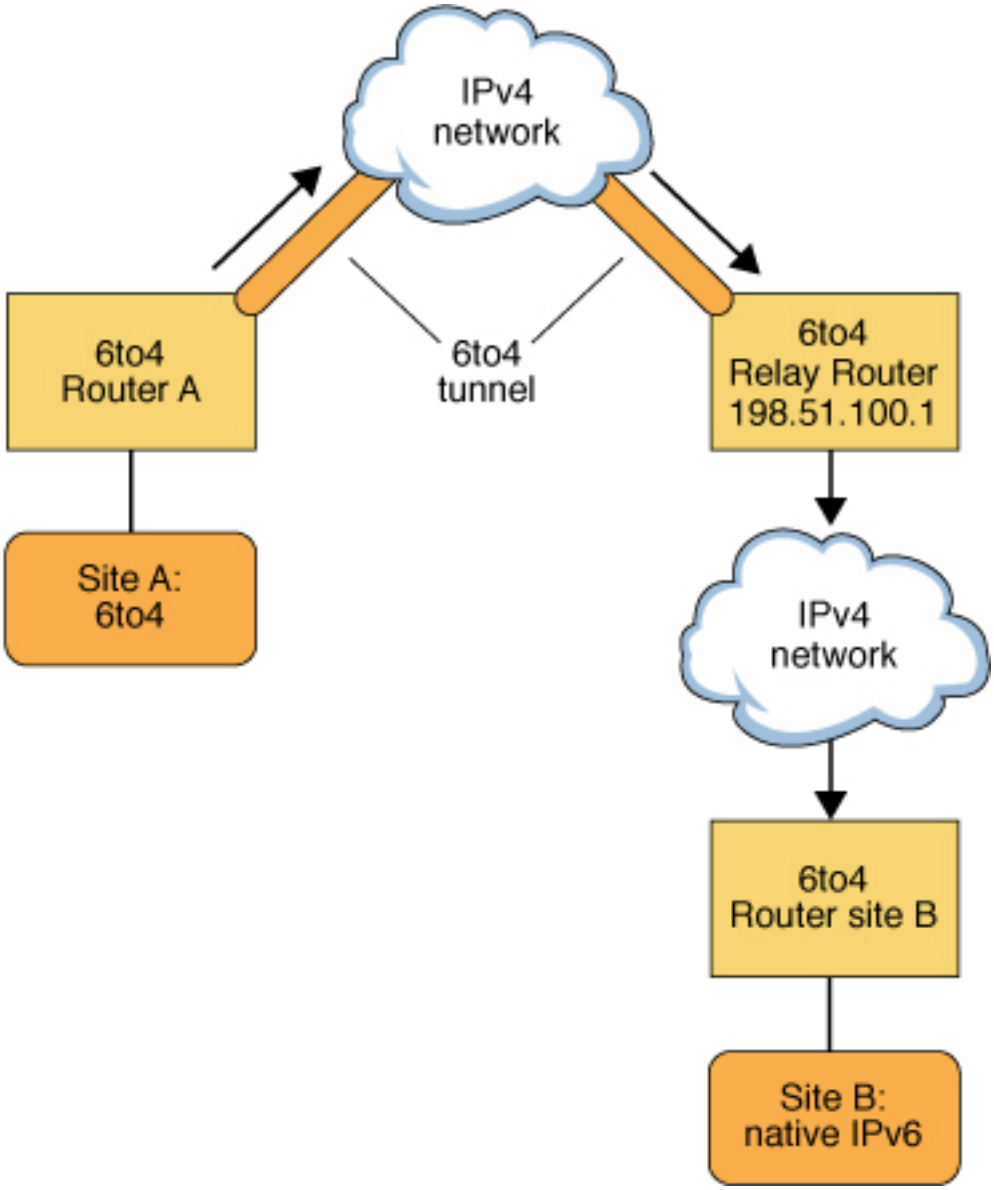
address. For each IPv6 packet that flows through the tunnel interface, the packet's IPv6 destination address also contains the IPv4 destination address. Thus, the router is able to determine the IPv4 destination address that is set on the encapsulating header. Then, the router uses standard IPv4 routing 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 105, 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

This section describes 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 105.

The packet flow is as follows:

1. A system on 6to4 Site A sends a transmission that specifies a system at native IPv6 Site B as the destination. 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 `198.51.100.1` anycast group.

Note - 6to4 relay routers that are part of the 6to4 relay router anycast group have the IP address `198.51.100.1`. 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 requirements need to be satisfied 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 in the Network" in *Planning for Network Deployment in Oracle Solaris 11.4*.

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 1 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.
	IPv6 interface	Local and remote link-local addresses are automatically set when you use the `ipadm create-addr -T addrconf` command. See the `ipadm`(8) man page.
IPv6 tunnel	IPv4 interface	Local and remote addresses are manually specified.

Tunnel Type	IP Interface Allowed Over Tunnel	IP Interface Requirement
6to4 tunnel	IPv6 interface	Local and remote link-local addresses are automatically set when you use the `ipadm create-addr -T addrconf` command. See the ipadm(8) man page.
	IPv6 interface only	Default IPv6 address is automatically selected when you use the `ipadm create-ip` command. See the ipadm(8) 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.

♦ ♦ ♦ CHAPTER 5

Administering IP Tunnels

This chapter describes tasks for administering IP tunnels in Oracle Solaris.

This chapter contains the following topics:

- "About IP Tunnel Administration in Oracle Solaris"
- "Administering IP Tunnels"

Note - To administer IP tunnels and issue commands described in this chapter, you must have the appropriate rights profile. See "Using Rights Profiles to Perform Network Configuration".

About IP Tunnel Administration in Oracle Solaris

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`

See the `dladm`(8) and `ipadm`(8) man pages.

Note - 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 6, "About IP Security Architecture" in *Securing the Network in Oracle Solaris 11.4*. To configure IPsec, see Chapter 7, "Configuring IPsec" in *Securing the Network in Oracle Solaris 11.4*.

Administering IP Tunnels

This section contains the following topics:

▼ How to Create and Configure an IP Tunnel

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

1. 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. 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 - If you are using host names for addresses for persistent IP tunnel data-link configurations, 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 network-link administration in this release, 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, see "Rules for Valid Link Names" in *Configuring and Managing Network Components in Oracle Solaris 11.4*.

2. (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`.

3. Create an IP interface over the tunnel.

```
$ ipadm create-ip tunnel-interface
```

where *tunnel-interface* uses the same name as the tunnel link.

4. 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.

See *Configuring and Managing Network Components in Oracle Solaris 11.4* and the `ipadm`(8) man page.

5. **(Optional) Verify the status of the tunnel's IP interface configuration.**

```
$ ipadm show-addr interface
```

Example 25 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 26 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=203.0.113.1,remote=203.0.113.2 vpn0
vpn0/v4
$ ipadm show-addr vpn0/
ADDROBJ           TYPE     STATE   ADDR
vpn0/v4           static   ok      203.0.113.1->203.0.113.2
```

You can further configure IPsec policy to provide secure connections for the packets that flow over this tunnel. For information, see Chapter 7, “Configuring IPsec” in *Securing the Network in Oracle Solaris 11.4*.

Example 27 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 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 host as a router, see “Configuring an IPv6 Router” in *Configuring an Oracle Solaris 11.4 System as a Router or a Load Balancer*.

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See “Using Rights Profiles to Perform Network Configuration” on page 13.

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. With support for meaningful names in a network-link administration, tunnel names are no longer restricted to the type of tunnel that you are creating. Instead, you can assign a tunnel any administratively-chosen name. Tunnel names consist of a string and the PPA number, for example, *mytunnel0*. For more information, see "Rules for Valid Link Names" in *Configuring and Managing Network Components in Oracle Solaris 11.4*.

2. **Create the tunnel IP interface.**

   ```
   $ ipadm create-ip tunnel-interface
   ```

   where *tunnel-interface* uses the same name as the tunnel link.

3. **(Optional) Add alternative IPv6 addresses for the tunnel's use.**

4. **To advertise 6to4 routing, add the following two lines to the `/etc/inet/ndpd.conf` file.**

   ```
   if subnet-interface AdvSendAdvertisements 1
   IPv6-address  subnet-interface
   ```

   where the first line specifies the subnet that receives the advertisement and the *subnet-interface* refers to the link to which the subnet is connected. The IPv6 address on the second line must have the 6to4 prefix `2000` that is used for IPv6 addresses in 6to4 tunnels.

   See the `ndpd.conf`(5) man page.

5. **Enable IPv6 forwarding.**

   ```
   $ ipadm set-prop -p forwarding=on ipv6
   ```

6. **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.

7. **Add the new 6to4-derived addresses of the nodes to the name service that is used at the 6to4 site.**

For instructions, see Chapter 4, "Administering Naming and Directory Services on an Oracle Solaris System" in *Configuring and Managing Network Components in Oracle Solaris 11.4*.

Example 28 Creating a 6to4 Tunnel

The following example shows how to 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 11.4* 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 110.
- Review the security issues that are involved in tunneling to a 6to4 relay router.

Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

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, see 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 that connects 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. **On the `/etc/default/inetinit`, change the `NO` value in the `ACCEPT6TO4RELAY=NO` line to `YES`.**

      Note that the line you modify is at the end of the file.

   b. **(Optional) Create a tunnel that connects 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 29 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 the configuration of a tunnel by using the following command syntax:

```
$ dladm modify-iptun -a [local|remote]=addr,... tunnel-link
```

You cannot modify the type of an existing tunnel. 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 data-link 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 30 Modifying Address and Properties of a Tunnel

The following example consists of two procedures. First, the local and remote addresses of the IPv4 tunnel `vpn0` are temporarily changed. When the system is later 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=203.0.113.149,remote=192.0.2.3 vpn0

$ dladm set-linkprop -p hoplimit=60 vpn0
```

Displaying the Configuration of an IP Tunnel

You display the configuration of an IP tunnel 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 in the system.

EXAMPLE 31 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     --       192.0.2.10/27   --
vpn0     ipv4     --       203.0.113.149   192.0.2.35/27
```

EXAMPLE 32 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
tun0:6to4:192.0.2.10
vpn0:ipv4:203.0.113.149
```

Displaying the Properties of an IP Tunnel

You display properties of an IP tunnel link 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 in the system.

EXAMPLE 33 Displaying a Tunnel's Properties

The following example shows how to display all the link properties of a tunnel.

```
$ 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

Before You Begin Ensure that your role has the appropriate rights profile to perform this procedure. See "Using Rights Profiles to Perform Network Configuration" on page 13.

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 34 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

L

M

N

U

V