```
**********************************************************
    6696 Fri Dec 19 04:18:42 2008
new/./arp.1m.txt
Clearview IPMP manpages
**********************************************************
```

```
   1 System Administration Commands                           arp(1M)



   5 NAME
   6      arp - address resolution display and control

   8 SYNOPSIS
   9      arp hostname


  12      arp -a [-n]


  15      arp -d hostname


  18      arp -f filename


  21      arp -s hostname ether_address [temp] [pub] [trail]
  22          [permanent]


  25 DESCRIPTION
  26      The arp program displays and  modifies  the  Internet-to-MAC
  27      address  translation  tables  used by the address resolution
  28      protocol (see arp(7P)).


  31      With no flags, the program displays the  current  ARP  entry
  32      for  hostname.  The  host  may  be  specified  by name or by
  33      number, using Internet dot notation.


  36      Options that modify the ARP translation tables (-d, -f,  and
  37      -s) can be used only when the invoked command is granted the
  38      sys_net_config privilege. See privileges(5).

  40 OPTIONS
  41      -a    Display all of the current ARP entries. The definition
  42            for the flags in the table are:

  44            d    Unverified; this is a local IP  address  that  is
  45                 currently undergoing Duplicate Address Detection.
  46                 ARP will not respond to requests for this address
  47                 until Duplicate Address Detection completes.


  50            o    Old; this entry is aging away. If IP requests  it
  51                 again,  a  new  ARP query will be generated. This
  52                 state is used for detecting peer address changes.


  55            y    Delayed; periodic address  defense  and  conflict
  56                 detection  was  unable  to  send  a packet due to


  60 SunOS 5.11          Last change: 25 Jul 2006                   1
```



```
  67 System Administration Commands                           arp(1M)


  71                 internal  network  use  limits  for  non-traffic-
  72                 related messages (100 packets per hour per inter-
  73                 face). This occurs only on interfaces  with  very
  74                 large numbers of aliases.


  77            A    Authority; this machine is authoritative for this
  78                 IP  address.  ARP  will  not  accept updates from
  79                 other machines for this entry.


  82            L    Local; this is a local IP address  configured  on
  83                 one of the machine's logical interfaces. ARP will
  84                 defend this address if another node  attempts  to
  85                 claim it.


  88            M    Mapping; only used for the  multicast  entry  for
  89                 224.0.0.0


  92            P    Publish; includes IP address for the machine  and
  93                 the  addresses that have explicitly been added by
  94                 the -s option. ARP will respond to  ARP  requests
  95                 for this address.


  98            S    Static; entry cannot be changed by learned infor-
  99                 mation.  This  indicates  that the permanent flag
 100                 was used when creating the entry.


 103            U    Unresolved; waiting for ARP response.

 105            You can use the -n option with the -a option  to  dis-
 106            able the automatic numeric IP address-to-name transla-
 107            tion. Use arp -an or arp -na  to  display  numeric  IP
 108            addresses. The arp -a option is equivalent to:

 110              # netstat -p -f inet


 113            ...and -an and -na are equivalent to:

 115              # netstat -pn -f inet




 120      -d    Delete an entry for the host called hostname.
 121
 122            Note that ARP entries for IPMP data and test addresses
 123            are managed by the kernel, and thus cannot be deleted.
```

139 -f Read the file named filename and set multiple entries
140 in the ARP tables. Entries in the file should be of
141 the form:

143 hostname MACaddress [temp] [pub] [trail] [permanent]

146 See the -s option for argument definitions.

149 -s Create an ARP entry for the host called hostname with
150 the MAC address MACaddress. For example, an Ethernet
151 address is given as six hexadecimal bytes separated by
152 colons. The entry will not be subject to deletion by
153 aging unless the word temp is specified in the com-
154 mand. If the word pub is specified, the entry will be
155 published, which means that this system will respond
156 to ARP requests for hostname even though the hostname
157 is not its own. The word permanent indicates that the
158 system will not accept MAC address changes for host-
159 name from the network.

161 Solaris does not implement trailer encapsulation, and
162 the word trail is accepted on entries for compatibil-
163 ity only.

165 arp -s can be used for a limited form of proxy ARP
166 when a host on one of the directly attached networks
167 is not physically present on a subnet. Another machine
168 can then be configured to respond to ARP requests
169 using arp -s. This is useful in certain SLIP confi-
170 gurations.

172 **Non-temporary proxy ARP entries for an IPMP group are**
173 **automatically managed by the kernel. Specifically, if**
174 **the hardware address in an entry matches the hardware**
175 **address of an IP interface in an IPMP group, and the**
176 **IP address is not local to the system, this will be**
177 **regarded as an IPMP proxy ARP entry. This entry will**
178 **have its hardware address automatically adjusted in**
179 **order to keep the IP address reachable so long as the**
180 **IPMP group has not entirely failed.**

182 **ARP entries must be consistent across an IPMP group.**
183 **Therefore, ARP entries cannot be associated with**
184 **individual underlying IP interfaces in an IPMP group,**
185 **and must instead be associated with the corresponding**
186 **IPMP IP interface.**

188 **Note that ARP entries for IPMP data and test addresses**
189 **are managed by the kernel, and thus cannot be changed.**

192 ATTRIBUTES
193 See attributes(5) for descriptions of the following attri-

194 butes:

198
199 | ATTRIBUTE TYPE | ATTRIBUTE VALUE |
200 |---|---|
201 | Availability | SUNWcsu |
202

205 SEE ALSO
206 ifconfig(1M), netstat(1M), attributes(5), privileges(5),
207 arp(7P)

```
***********************************************************
   13679 Fri Dec 19 04:18:44 2008
new/./arp.7p.txt
Clearview IPMP manpages
***********************************************************
```

```
NAME
     arp, ARP - Address Resolution Protocol

SYNOPSIS
     #include <sys/fcntl.h>

     #include <sys/socket.h>

     #include <net/if_arp.h>

     #include <netinet/in.h>

     s = socket(AF_INET, SOCK_DGRAM, 0);

     d = open ("/dev/arp", oflag);

DESCRIPTION
     ARP is a protocol used to map  dynamically between  Internet
     Protocol (IP) and Ethernet addresses. It is used by all Eth-
     ernet datalink providers (network drivers) and can  be  used
     by  other datalink providers that support broadcast, includ-
     ing FDDI  and Token Ring. The  only network layer  supported
     in  this  implementation  is the Internet Protocol, although
     ARP is not specific to that protocol.

     ARP caches IP-to-link-layer address mappings. When an inter-
     face requests a mapping for an address not in the cache, ARP
     queues the message that requires the mapping and  broadcasts
     a  message  on the associated network requesting the address
     mapping. If a response is provided, ARP caches the new  map-
     ping  and  transmits  any  pending message. ARP will queue a
     maximum of four packets while awaiting a response to a  map-
     ping   request.  ARP  keeps  only the first four transmitted
     packets.

APPLICATION PROGRAMMING INTERFACE
     The STREAMS device /dev/arp is not a Transport Level  Inter-
     face  (TLI)  transport provider and may not be used with the
     TLI interface.

     To facilitate  communications with  systems that do  not use
     ARP,   ioctl()  requests are  provided  to  enter and delete
     entries  in  the  IP-to-link address  tables.  Ioctls   that
     change the table  contents require sys_net_config privilege.

SunOS 5.10           Last change: 2 Dec 2008                   1

Protocols                                                arp(7P)

     See privileges(5).

       #include <sys/sockio.h>
       #include <sys/socket.h>
       #include <net/if.h>
       #include <net/if_arp.h>
       struct arpreq arpreq;
       ioctl(s, SIOCSARP, (caddr_t)&arpreq);
       ioctl(s, SIOCGARP, (caddr_t)&arpreq);
       ioctl(s, SIOCDARP, (caddr_t)&arpreq);

     SIOCSARP, SIOCGARP and SIOCDARP are BSD  compatible  ioctls.
```



```
     These  ioctls  do  not  communicate  the  mac address length
     between the user and the kernel (and thus only  work  for  6
     byte  wide  Ethernet addresses). To manage the ARP cache for
     media that has different sized mac addresses, use SIOCSXARP,
     SIOCGXARP and SIOCDXARP ioctls.

       #include <sys/sockio.h>
       #include <sys/socket.h>
       #include <net/if.h>
       #include <net/if_dl.h>
       #include <net/if_arp.h>
       struct xarpreq xarpreq;
       ioctl(s, SIOCSXARP, (caddr_t)&xarpreq);
       ioctl(s, SIOCGXARP, (caddr_t)&xarpreq);
       ioctl(s, SIOCDXARP, (caddr_t)&xarpreq);

     Each ioctl() request takes the same structure  as  an  argu-
     ment. SIOCS[X]ARP sets an ARP entry, SIOCG[X]ARP gets an ARP
     entry, and SIOCD[X]ARP deletes an ARP entry.  These  ioctl()
     requests  may  be  applied  to  any  Internet  family socket
     descriptors, or to a descriptor for  the  ARP  device.  Note
     that  SIOCS[X]ARP and SIOCD[X]ARP require a privileged user,
     while SIOCG[X]ARP does not.

     The arpreq structure contains

       /*
       * ARP ioctl request
       */
       struct arpreq {
          struct sockaddr arp_pa;  /* protocol address */
          struct sockaddr arp_ha; /* hardware address */
          int  arp_flags;         /* flags */
       };

SunOS 5.10           Last change: 2 Dec 2008                   2

Protocols                                                arp(7P)

     The xarpreq structure contains:

       /*
       * Extended ARP ioctl request
       */
       struct xarpreq {
          struct sockaddr_storage xarp_pa; /* protocol address */
          struct sockaddr_dl xarp_ha;     /* hardware address */
          int xarp_flags;                 /* arp_flags field values */
       };
       #define ATF_COM 0x2          /* completed entry (arp_ha valid) */
       #define ATF_PERM 0x4         /* permanent (non-aging) entry */
       #define ATF_PUBL 0x8         /* publish (respond for other host) */
       #define ATF_USETRAILERS 0x10 /* send trailer pckts to host */
       #define ATF_AUTHORITY 0x20   /* hardware address is authoritative */

     The address  family  for  the  [x]arp_pa  sockaddr  must  be
     AF_INET.  The  ATF_COM  flag  bits  ([x]arp_flags) cannot be
     altered.  ATF_USETRAILERS is not implemented on Solaris  and
     altered.  ATF_USETRAILER  is not implemented on Solaris  and
     is  retained   for  compatibility  only.  ATF_PERM makes the
     entry permanent (disables aging)   if  the  ioctl()  request
     succeeds.  ATF_PUBL specifies that the system should respond
     to ARP requests for the indicated  protocol  address  coming
     from  other  machines.  This allows a host to act as an "ARP
     server," which may  be  useful  in  convincing  an  ARP-only
     machine  to  talk to a non-ARP  machine. ATF_AUTHORITY indi-
```

cates that this machine owns the address. ARP does not update the entry based on received packets.

The address family for the arp_ha sockaddr must be AF_UNSPEC.

Before invoking any of the SIOC*XARP ioctls, user code must fill in the xarp_pa field with the protocol (IP) address information, similar to the BSD variant. The SIOC*XARP ioctls come in two (legal) varieties, depending on xarp_ha.sdl_nlen:

1. if sdl_nlen = 0, it behaves as an extended BSD ioctl. The kernel uses the IP address to determine the network interface.
2. if (sdl_nlen > 0) and (sdl_nlen < LIFNAMSIZ), the kernel uses the interface name in sdl_data[0] to determine the network interface; sdl_nlen represents the length of the string (excluding terminating null character).
3. if (sdl_nlen >= LIFNAMSIZ), an error (EINVAL) is flagged from the ioctl.

Other than the above, the xarp_ha structure should be 0-filled except for SIOCSXARP, where the sdl_alen field must be set to the size of hardware address length and the hardware address itself must be placed in the LLADDR/sdl_data[] area. (EINVAL will be returned if user specified sdl_alen does not match the address length of the identified interface).

On return from the kernel on a SIOCGXARP ioctl, the kernel fills in the name of the interface (excluding terminating NULL) and its hardware address, one after another, in the sdl_data/LLADDR area; if the two are larger than can be held in the 244 byte sdl_data[] area, an ENOSPC error is returned. Assuming it fits, the kernel will also set sdl_alen with the length of hardware address, sdl_nlen with the length of name of the interface (excluding terminating NULL), sdl_type with an IFT_* value to indicate the type of the media, sdl_slen with 0, sdl_family with AF_LINK and sdl_index (which if not 0) with system given index for the interface. The information returned is very similar to that returned via routing sockets on an RTM_IFINFO message.

The ARP ioctls have several additional restrictions and enhancements when used in conjunction with IPMP:

- **The ARP mappings for IPMP data and test addresses are managed by the kernel, and thus cannot be changed through any ARP ioctls, though they may be retrieved using *SIOCGARP* or *SIOCGXARP*.**
- **ARP mappings for a given IPMP group must be consistent across the group. Therefore, ARP mappings cannot be associated with individual underlying IP interfaces in an IPMP group, and must instead be associated with the corresponding IPMP IP interface.**
- **Proxy ARP mappings for an IPMP group are automatically managed by the kernel. Specifically, if the hardware address in a *SIOCSARP* or *SIOCSXARP* request matches the hardware address of an IP interface in an IPMP group, and the IP address is not local to the system,**

the kernel will regard this as a IPMP Proxy ARP entry. This IPMP Proxy ARP entry will have its hardware address automatically adjusted in order to keep the IP address reachable (provided the IPMP group has not entirely failed).

ARP performs duplicate address detection for local addresses. When a logical interface is brought up (IFF_UP) or any time the hardware link goes up (IFF_RUNNING), ARP sends probes (ar$spa == 0) for the assigned address. If a conflict is found, the interface is torn down. See ifconfig(1M) for more details.

ARP watches for hosts impersonating the local host, that is, any host that responds to an ARP request for the local host's address, and any address for which the local host is an authority. ARP defends local addresses and logs those with ATF_AUTHORITY set, and can tear down local addresses on an excess of conflicts.

ARP also handles UNARP messages received from other nodes. It does not generate these messages.

PACKET EVENTS

The arp driver registers itself with the netinfo interface. To gain access to these events, a handle from net_protocol_lookup must be acquired by passing it the value NHF_ARP. Through this interface, two packet events are supported:

Physical in - ARP packets received via a network inter face

Physical out - ARP packets to be sent out via a network interface

For ARP packets, the hook_pkt_event structure is filled out as follows:

hpe_ifp

Identifier indicating the inbound interface for packets received with the "physical in" event.

hpe_ofp

Identifier indicating the outbound interface for packets received with the "physical out" event.

hpe_hdr

Pointer to the start of the ARP header (not the ethernet header).

hpe_mp

Pointer to the start of the mblk_t chain containing the ARP packet.

hpe_mb

Pointer to the mblk_t with the ARP header in it.

```
NETWORK INTERFACE EVENTS
     In addition  to events  describing  packets  as  they   move
     through the system, it is also possible to receive notifica-
     tion of events relating to network interfaces. These  events
     are   all  reported back through the same callback. The list
     of events is as follows:

     plumb

         A new network interface has been instantiated.

SunOS 5.10           Last change: 2 Dec 2008                   5

Protocols                                                arp(7P)

     unplumb

         A network interface is no longer  associated with ARP.

SEE ALSO
     arp(1M), ifconfig(1M), privileges(5), if_tcp(7P),  inet(7P),
     netinfo(9F)

     Plummer, Dave, An Ethernet Address  Resolution  Protocol  or
     Converting  Network  Protocol  Addresses to 48 .bit Ethernet
     Addresses for Transmission on Ethernet  Hardware, RFC   826,
     STD 0037, November 1982.

     Malkin,  Gary, ARP  Extension  - UNARP, RFC 1868,  November,
     1995

DIAGNOSTICS
     Several messages can be written to the system  logs (by  the
     IP module) when errors occur. In the following examples, the
     hardware address strings include colon (:)  separated  ASCII
     representations  of  the link layer addresses, whose lengths
     depend on the underlying media (for  example,  6  bytes  for
     Ethernet).

     Node %x:%x ... %x:%x is using our IP address %d.%d.%d.%d on
     %s.

         Duplicate IP address warning. ARP has discovered another
         host  on  a  local  network  that  responds  to  mapping
         requests for the Internet  address of this  system,  and
         has  defended  the  system  against  this  node  by  re-
         announcing the ARP entry.

     %s has duplicate address %d.%d.%d.%d (in use by %x:%x ...
     %x:%x); disabled.

         Duplicate IP address detected while  performing  initial
         probing.  The   newly-configured interface has been shut
         down.

     %s has duplicate address %d.%d.%d.%d (claimed by %x:%x ...
     %x:%x); disabled.

         Duplicate IP address detected on a running IP interface.
         The  conflict  cannot be resolved, and the interface has
         been disabled to protect the network.

SunOS 5.10           Last change: 2 Dec 2008                   6

Protocols                                                arp(7P)
```



```
     Recovered address %d.%d.%d.%d on %s.

         An interface with a  previously-conflicting  IP  address
         has been recovered automatically and reenabled. The con-
         flict has been resolved.

     Proxy ARP problem?  Node '%x:%x ... %x:%x' is using
     %d.%d.%d.%d on %s

         This  message appears if arp(1M) has been used to create
         a  published  permanent  (ATF_AUTHORITY) entry, and some
         other host on the  local  network  responds  to  mapping
         requests for the published ARP entry.
```

****
26006 Fri Dec 19 04:18:45 2008
new/./dhcpagent.txt
Clearview IPMP manpages
****

System Administration Commands dhcpagent(1M)

NAME

dhcpagent - Dynamic Host Configuration Protocol (DHCP) client daemon

SYNOPSIS

dhcpagent [-a] [-d n] [-f] [-v]

DESCRIPTION

dhcpagent implements the client half of the Dynamic Host Configuration Protocol (DHCP) for machines running Solaris software.

The dhcpagent daemon obtains configuration parameters for the client (local) machine's network interfaces from a DHCP server. These parameters may include a lease on an IP address, which gives the client machine use of the address for the period of the lease, which may be infinite. If the client wishes to use the IP address for a period longer than the lease, it must negotiate an extension using DHCP. For this reason, dhcpagent must run as a daemon, terminating only when the client machine powers down.

For IPv4, the dhcpagent daemon is controlled through ifconfig(1M) in much the same way that the init(1M) daemon is controlled by telinit(1M). dhcpagent can be invoked as a user process, albeit one requiring root privileges, but this is not necessary, as ifconfig(1M) will start it automatically.

For IPv6, the dhcpagent daemon is invoked automatically by in.ndpd(1M). It can also be controlled through ifconfig(1M), if necessary.

When invoked, dhcpagent enters a passive state while it awaits instructions from ifconfig(1M) or in.ndpd(1M). When it receives a command to configure an interface, it brings up the interface (if necessary) and starts DHCP. Once DHCP is complete, dhcpagent can be queried for the values of the various network parameters. In addition, if DHCP was used to obtain a lease on an address for an interface, it configures the address for use. When a lease is obtained, it is automatically renewed as necessary. If the lease cannot be renewed, dhcpagent will unconfigure the address, but the interface will be left up and dhcpagent will attempt to acquire a new address lease. dhcpagent monitors system suspend/resume events and will validate any non-permanent leases with the DHCP server upon resume. Similarly,

dhcpagent monitors link up/down events and will validate any non-permanent leases with the DHCP server when the downed link is brought back up.

For IPv4, if the configured interface is found to be **unplumbed, or to have a different IP address, subnet mask, or broadcast address from those obtained from DHCP, the interface is abandoned from DHCP control.**
unplumbed, marked down, or to have a different IP address, subnet mask, or broadcast address from those obtained from DHCP, the interface is abandoned by DHCP control.

For IPv6, dhcpagent automatically plumbs and unplumbs logical interfaces as necessary for the IPv6 addresses supplied by the server. The IPv6 prefix length (netmask) is not set by the DHCPv6 protocol, but is instead set by *in.ndpd(1M)* using prefix information obtained by Router Advertisements. If any of the logical interfaces created by dhcpagent is unplumbed, or configured with a different IP address, it will be abandoned from DHCP control. If the link-local interface is unplumbed, then all addresses configured by DHCP on that physical interface will be removed.
For IPv6, dhcpagent automatically plumbs and unplumbs logical interfaces as necessary for the IPv6 addresses supplied by the server. The IPv6 prefix length (netmask) is not set by the DHCPv6 protocol, but is instead set by in.ndpd(1M) using prefix information obtained by Router Advertisements. If any of the logical interfaces created by dhcpagent is unplumbed, marked down, or configured with a different IP address, it will be abandoned by DHCP control. If the link-local interface is unplumbed, then all addresses configured by DHCP on that physical interface will be removed.

In addition to DHCP, dhcpagent also supports BOOTP (IPv4 only). See RFC 951, Bootstrap Protocol. Configuration parameters obtained from a BOOTP server are treated identically to those received from a DHCP server, except that the IP address received from a BOOTP server always has an infinite lease.

DHCP also acts as a mechanism to configure other information needed by the client, for example, the domain name and addresses of routers. Aside from the IP address, and for IPv4 alone, the netmask, broadcast address, and default router, the agent does not directly configure the workstation, but instead acts as a database which may be interrogated by other programs, and in particular by dhcpinfo(1).

On clients with a single interface, this is quite straightforward. Clients with multiple interfaces may present difficulties, as it is possible that some information arriving on

different interfaces may need to be merged, or may be inconsistent. Furthermore, the configuration of the interfaces is asynchronous, so requests may arrive while some or all of the interfaces are still unconfigured. To handle these cases, one interface may be designated as primary, which makes it the authoritative source for the values of DHCP parameters in the case where no specific interface is requested. See dhcpinfo(1) and ifconfig(1M) for details.

For IPv4, the dhcpagent daemon can be configured to request a particular host name. See the REQUEST_HOSTNAME description in the FILES section. When first configuring a client to request a host name, you must perform the following steps as root to ensure that the full DHCP negotiation takes place:

```
# pkill dhcpagent
# rm /etc/dhcp/interface.dhc
# reboot
```

All DHCP packets sent by dhcpagent include a vendor class identifier (RFC 2132, option code 60; RFC 3315, option code 16). This identifier is the same as the platform name returned by the uname -i command, except:

- Any commas in the platform name are changed to periods.
- If the name does not start with a stock symbol and a comma, it is automatically prefixed with SUNW.

Messages

The dhcpagent daemon writes information and error messages in five categories:

critical

Critical messages indicate severe conditions that prevent proper operation.

errors

Error messages are important, sometimes unrecoverable events due to resource exhaustion and other unexpected failure of system calls; ignoring errors may lead to degraded functionality.

warnings

Warnings indicate less severe problems, and in most cases, describe unusual or incorrect datagrams received from servers, or requests for service that cannot be provided.

informational

Informational messages provide key pieces of information that can be useful to debugging a DHCP configuration at a site. Informational messages are generally controlled by the -v option. However, certain critical pieces of information, such as the IP address obtained, are always provided.

debug

Debugging messages, which may be generated at two different levels of verbosity, are chiefly of benefit to persons having access to source code, but may be useful as well in debugging difficult DHCP configuration problems. Debugging messages are only generated when using the -d option.

When dhcpagent is run without the -f option, all messages are sent to the system logger syslog(3C) at the appropriate matching priority and with a facility identifier LOG_DAEMON. When dhcpagent is run with the -f option, all messages are directed to standard error.

DHCP Events and User-Defined Actions

If an executable (binary or script) is placed at /etc/dhcp/eventhook, the dhcpagent deamon will automatically run that program when any of the following events occur:

BOUND and BOUND6

These events occur during interface configuration. The event program is invoked when dhcpagent receives the DHCPv4 ACK or DHCPv6 Reply message from the DHCP server for the lease request of an address, indicating successful initial configuration of the interface. (See also the INFORM and INFORM6 events, which occur when configuration parameters are obtained without address leases.)

EXTEND and EXTEND6

These events occur during lease extension. The event program is invoked just after dhcpagent receives the DHCPv4 ACK or DHCPv6 Reply from the DHCP server for the DHCPv4 REQUEST (renew) message or the DHCPv6 Renew or Rebind message.

Note that with DHCPv6, the server might choose to remove some addresses, add new address leases, and ignore

(allow to expire) still other addresses in a given Reply message. The EXTEND6 event occurs when a Reply is received that leaves one or more address leases still valid, even if the Reply message does not extend the lease for any address. The event program is invoked just before any addresses are removed, but just after any new addresses are added. Those to be removed will be marked with the IFF_DEPRECATED flag.

EXPIRE and EXPIRE6

These events occur during lease expiration. For DHCPv4, the event program is invoked just before the leased address is removed from an interface and the interface is marked as down. For DHCPv6, the event program is invoked just before the last remaining leased addresses are removed from the interface.

DROP and DROP6

These events occur during the period when an interface is dropped. The event program is invoked just before the interface is removed from DHCP control. If the interface has been abandoned due the user unplumbing the interface, then this event will occur after the user's action has taken place. The interface might not be present.

INFORM and INFORM6

These events occur when an interface acquires new or updated configuration information from a DHCP server by means of the DHCPv4 INFORM or the DHCPv6 Information-Request message. These messages are sent using an ifconfig(1M) dhcp inform command or when the DHCPv6 Router Advertisement O (letter 0) bit is set and the M bit is not set. Thus, these events occur when the DHCP client does not obtain an IP address lease from the server, and instead obtains only configuration parameters.

LOSS6

This event occurs during lease expiration when one or more valid leases still remain. The event program is invoked just before expired addresses are removed. Those being removed will be marked with the IFF_DEPRECATED flag.

Note that this event is not associated with the receipt of the Reply message, which occurs only when one or more valid leases remain, and occurs only with DHCPv6. If all leases have expired, then the EXPIRE6 event occurs instead.

RELEASE and RELEASE6

This event occurs during the period when a leased address is released. The event program is invoked just before dhcpagent relinquishes the address on an interface and sends the DHCPv4 RELEASE or DHCPv6 Release packet to the DHCP server.

The system does not provide a default event program. The file /etc/dhcp/eventhook is expected to be owned by root and have a mode of 755.

The event program will be passed two arguments, the interface name and the event name, respectively. For DHCPv6, the interface name is the name of the physical interface.

The event program can use the dhcpinfo(1) utility to fetch additional information about the interface. While the event program is invoked on every event defined above, it can ignore those events in which it is not interested. The event program runs with the same privileges and environment as dhcpagent itself, except that stdin, stdout, and stderr are redirected to /dev/null. Note that this means that the event program runs with root privileges.

If an invocation of the event program does not exit after 55 seconds, it is sent a SIGTERM signal. If does not exit within the next three seconds, it is terminated by a SIGKILL signal.

See EXAMPLES for an example event program.

OPTIONS

The following options are supported:

-a

Adopt a configured IPv4 interface. This option is for use with diskless DHCP clients. In the case of diskless

DHCP, DHCP has already been performed on the network interface providing the operating system image prior to running dhcpagent. This option instructs the agent to take over control of the interface. It is intended primarily for use in boot scripts.

The effect of this option depends on whether the interface is being adopted.

If the interface is being adopted, the following conditions apply:

dhcpagent uses the client id specified in /chosen:<client_id>, as published by the PROM or as specified on a boot(1M) command line. If this value is not present, the client id is undefined. The DHCP server then determines what to use as a client id. It is an error condition if the interface is an Infiniband interface and the PROM value is not present.

If the interface is not being adopted:

dhcpagent uses the value stored in /etc/default/dhcpagent. If this value is not present, the client id is undefined. If the interface is Infiniband and there is no value in /etc/default/dhcpagent, a client id is generated as described by the draft document on DHCP over Infiniband, available at:

http://www.ietf.org

-d n

Set debug level to n. Two levels of debugging are currently available, 1 and 2; the latter is more verbose.

-f

Run in the foreground instead of as a daemon process.

When this option is used, messages are sent to standard error instead of to syslog(3C).

-v

Provide verbose output useful for debugging site configuration problems.

EXAMPLES

Example 1 Example Event Program

The following script is stored in the file /etc/dhcp/eventhook, owned by root with a mode of 755. It is invoked upon the occurrence of the events listed in the file.

```
#!/bin/sh

(
echo "Interface name: " $1
echo "Event: " $2

case $2 in
"BOUND")
    echo "Address acquired from server "\
        `/sbin/dhcpinfo -i $1 ServerID`
    ;;
"BOUND6")
    echo "Addresses acquired from server " \
        `/sbin/dhcpinfo -v6 -i $1 ServerID`
    ;;
"EXTEND")
   echo "Lease extended for " \
        `sbin/dhcpinfo -i $1 LeaseTim`" seconds"
    ;;
"EXTEND6")
   echo "New lease information obtained on $i"
    ;;
"EXPIRE" | "DROP" | "RELEASE")
    ;;

esac
) >/var/run/dhcp_eventhook_output 2>&1
```

Note the redirection of stdout and stderr to a file.

FILES
/etc/dhcp/if.dhc
/etc/dhcp/if.dh6

Contains the configuration for interface. The mere existence of this file does not imply that the configuration is correct, since the lease might have expired. On start-up, dhcpagent confirms the validity of the

address using REQUEST (for DHCPv4) or Confirm (DHCPv6).

/etc/dhcp/duid
/etc/dhcp/iaid

Contains persistent storage for DUID (DHCP Unique Identifier) and IAID (Identity Association Identifier) values. The format of these files is undocumented, and applications should not read from or write to them.

/etc/default/dhcpagent

Contains default values for tunable parameters. All values may be qualified with the interface they apply to by prepending the interface name and a period (".") to the interface parameter name. The parameters include: the interface parameter name.

To configure IPv6 parameters, place the string .v6 between the interface name (if any) and the parameter name. For example, to set the global IPv6 parameter request list, use .v6.PARAM_REQUEST_LIST. To set the CLIENT_ID (DUID) on hme0, use hme0.v6.CLIENT_ID.

The parameters include:

RELEASE_ON_SIGTERM

Indicates that a RELEASE rather than a DROP should be performed on managed interfaces when the agent terminates. Release causes the client to discard the lease, and the server to make the address available again. Drop causes the client to record the lease in /etc/dhcp/interface.dhc or /etc/dhcp/interface.dh6 for later use.

OFFER_WAIT

Indicates how long to wait between checking for valid OFFERs after sending a DISCOVER. For DHCPv6,

sets the time to wait between checking for valid Advertisements after sending a Solicit.

CLIENT_ID

Indicates the value that should be used to uniquely identify the client to the server. This value can take one of three basic forms:

decimal,data...
0xHHHHH...
"string...."

The first form is an RFC 3315 DUID. This is legal for both IPv4 DHCP and DHCPv6. For IPv4, an RFC 4361 Client ID is constructed from this value. In this first form, the format of data... depends on the decimal value. The following formats are defined for this first form:

1,hwtype,time,lla

Type 1, DUID-LLT. The hwtype value is an integer in the range 0-65535, and indicates the type of hardware. The time value is the number of seconds since midnight, January 1st, 2000 UTC, and can be omitted to use the current system time. The lla value is either a colon-separated MAC address or the name of a physical interface. If the name of an interface is used, the hwtype value can be omitted. For example: 1,,,hme0

2,enterprise,hex...

Type 2, DUID-EN. The enterprise value is an integer in the range 0-4294967295 and represents the SMI Enterprise number for an organization. The hex string is an even-length sequence of hexadecimal digits.

3,hwtype,lla

Type 3, DUID-LL. This is the same as DUID-LLT (type 1), except that a time stamp is not used.

*,hex

Any other type value (0 or 4-65535) can be used

with an even-length hexadecimal string.

The second and third forms of CLIENT_ID are legal for IPv4 only. These both represent raw Client ID (without RFC 4361), in hex, or NVT ASCII string format. Thus, Sun and 0x53756E are equivalent.

PARAM_REQUEST_LIST

Specifies a list of comma-separated integer values of options for which the client would like values.

REQUEST_HOSTNAME

Indicates the client requests the DHCP server to map the client's leased IPv4 address to the host name associated with the network interface that performs DHCP on the client. The host name must be specified in the /etc/hostname.interface file for the relevant interface on a line of the form

inet hostname

where hostname is the host name requested.

This option works with DHCPv4 only.

/etc/dhcp/eventhook

Location of a DHCP event program.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr
Interface Stability	Committed

SEE ALSO

dhcpinfo(1), ifconfig(1M), init(1M), in.ndpd(1M), syslog(3C), attributes(5), dhcp(5)

Croft, B. and Gilmore, J.,Bootstrap Protocol (BOOTP)RFC 951, Network Working Group, September 1985.

Droms, R., Dynamic Host Configuration Protocol, RFC 2131, Network Working Group, March 1997.

Lemon, T. and B. Sommerfeld. RFC 4361, Node-specific Client Identifiers for Dynamic Host Configuration Protocol Version Four (DHCPv4). Nominum and Sun Microsystems. February 2006.

Droms, R. RFC 3315, Dynamic Host Configuration Protocol for IPv6 (DHCPv6). Cisco Systems. July 2003.

NOTES

DHCP can be performed on IPv4 logical interfaces just as with physical interfaces. When used on a logical interface, the daemon automatically constructs a Client ID value based on the DUID and IAID values, according to RFC 4361. The */etc/default/dhcpclient* *CLIENT_ID* value, if any,
The dhcpagent daemon can be used on IPv4 logical interfaces, just as with physical interfaces. When used on a logical interface, the daemon automatically constructs a Client ID value based on the DUID and IAID values, according to RFC 4361. The /etc/default/dhcpclient CLIENT_ID value, if any,
overrides this automatic identifier.

As with physical IPv4 interfaces, the /etc/hostname.hme0:1 and /etc/dhcp.hme0:1 files must also be created in order for hme0:1 to be automatically plumbed and configured at boot. In addition, unlike physical IPv4 interfaces, dhcpagent does not add or remove default routes associated with logical interfaces.

DHCP can be performed on IPMP IP interfaces to acquire and maintain IPMP data addresses. Because an IPMP IP interface has no hardware address, the daemon automatically constructs a Client ID using the same approach described above for IPv4 logical interfaces. In addition, the lack of a hardware address means the daemon must set the "broadcast" flag in all *DISCOVER* and *REQUEST* messages on IPMP IP interfaces. Some DHCP servers may refuse such requests.

DHCP can be performed on IP interfaces that are part of an IPMP group (to acquire and maintain test addresses). The daemon will automatically set the *NOFAILOVER* and *DEPRECATED* flags on each test address. Additionally, the daemon will not add or remove default routes in this case. Note that the actual DHCP packet exchange may be performed over any active IP interface in the IPMP group. It is strongly recommended that test addresses have infinite leases. Otherwise, an extended network outage detectable only by probes may cause test address leases to expire, causing *in.mpathd(1M)* to revert to link-based failure detection and trigger an erroneous repair.

With DHCPv6, the link-local interface must be configured using /etc/hostname6.hme0 in order for DHCPv6 to run on hme0 at boot time. The logical interfaces for each address are plumbed by dhcpagent automatically.

****
14328 Fri Dec 19 04:18:47 2008
new/./getsockopt.3socket.txt
Clearview IPMP manpages
****

Sockets Library Functions getsockopt(3SOCKET)

NAME

getsockopt, setsockopt - get and set options on sockets

SYNOPSIS

```
cc [ flag ... ] file ... -lsocket  -lnsl  [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>

int getsockopt(int s, int level, int optname, void *optval,
    int *optlen);


int setsockopt(int s, int level, int optname, const void *optval,
    int optlen);
```

DESCRIPTION

The getsockopt() and setsockopt() functions manipulate options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost "socket" level.

The *level* argument specifies the protocol level at which the option resides. To manipulate options at the socket level, specify the *level* argument as *SOL_SOCKET*. To manipulate options at the protocol level, supply the appropriate protocol number for the protocol controlling the option. For example, to indicate that an option will be interpreted by TCP, set *level* to the protocol number of TCP, as defined in the *<netinet/in.h>* header, or as determined by using *getprotobyname(3SOCKET)*. Some socket protocol families may also define additional levels, such as *SOL_ROUTE*. Only socket-level options are described here.
When manipulating socket options, the level at which the option resides and the name of the option must be specified. To manipulate options at the "socket" level, level is specified as SOL_SOCKET. To manipulate options at any other level, level is the protocol number of the protocol that controls the option. For example, to indicate that an option is to be interpreted by the TCP protocol, level is set to the TCP protocol number. See getprotobyname(3SOCKET).

The parameters optval and optlen are used to access option values for setsockopt(). For getsockopt(), they identify a buffer in which the value(s) for the requested option(s) are to be returned. For getsockopt(), optlen is a value-result parameter, initially containing the size of the buffer pointed to by optval, and modified on return to indicate the actual size of the value returned. Use a 0 optval if no option value is to be supplied or returned.

The optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file <sys/socket.h> contains definitions

for the socket-level options described below. Options at other protocol levels vary in format and name.

Most socket-level options take an int for optval. For setsockopt(), the optval parameter should be non-zero to enable

Sockets Library Functions getsockopt(3SOCKET)

a boolean option, or zero if the option is to be disabled. SO_LINGER uses a struct linger parameter that specifies the desired state of the option and the linger interval. struct linger is defined in <sys/socket.h>. struct linger contains the following members:

```
l_onoff     on = 1/off = 0


l_linger    linger time, in seconds
```

The following options are recognized at the socket level. Except as noted, each may be examined with getsockopt() and set with setsockopt().

SO_DEBUG	enable/disable recording of debugging information
SO_REUSEADDR	enable/disable local address reuse
SO_KEEPALIVE	enable/disable keep connections alive
SO_DONTROUTE	enable/disable routing bypass for outgoing messages
SO_LINGER	linger on close if data is present
SO_BROADCAST	enable/disable permission to transmit broadcast messages
SO_OOBINLINE	enable/disable reception of out-of-band data in band
SO_SNDBUF	set buffer size for output
SO_RCVBUF	set buffer size for input

SO_DGRAM_ERRIND application wants delayed error

SO_TIMESTAMP enable/disable reception of timestamp with datagrams

SO_EXCLBIND enable/disable exclusive binding of the socket

SO_TYPE get the type of the socket (get only)

SO_ERROR get and clear error on the socket (get only)

SO_MAC_EXEMPT get or set mandatory access control on the socket. This option is available only when the system is configured with Trusted Extensions.

SO_ALLZONES bypass zone boundaries (privileged).

SO_DOMAIN get the domain used in the socket (get only)

SO_PROTOTYPE for socket in domains PF_INET and PF_INET6, get the underlying protocol number used in the socket. For socket in domain PF_ROUTE, get the address family used in the socket.

The SO_DEBUG option enables debugging in the underlying protocol modules. The SO_REUSEADDR option indicates that the rules used in validating addresses supplied in a bind(3SOCKET) call should allow reuse of local addresses. The SO_KEEPALIVE option enables the periodic transmission of messages on a connected socket. If the connected party fails to respond to these messages, the connection is considered broken and threads using the socket are notified using a SIGPIPE signal. The SO_DONTROUTE option indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

The SO_LINGER option controls the action taken when unsent messages are queued on a socket and a close(2) is performed. If the socket promises reliable delivery of data and SO_LINGER is set, the system will block the thread on the close() attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the setsockopt() call when SO_LINGER is requested). If SO_LINGER is disabled and a close() is issued, the system will process the close() in a manner that allows the thread to continue as quickly as possible.

The option SO_BROADCAST requests permission to send broadcast datagrams on the socket. With protocols that support out-of-band data, the SO_OOBINLINE option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with recv() or read() calls without the MSG_OOB flag.

The SO_SNDBUF and SO_RCVBUF options adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections or may be decreased to limit the possible backlog of incoming data. The maximum buffer size for UDP is determined by the value of the ndd variable udp_max_buf. The maximum buffer size for TCP is determined the value of the ndd variable tcp_max_buf. Use the ndd(1M) utility to determine the current default values. See the Solaris Tunable Parameters Reference Manual for information on setting the values of udp_max_buf and tcp_max_buf. At present, lowering SO_RCVBUF on a TCP connection after it has been established has no effect.

By default, delayed errors (such as ICMP port unreachable packets) are returned only for connected datagram sockets. The SO_DGRAM_ERRIND option makes it possible to receive errors for datagram sockets that are not connected. When this option is set, certain delayed errors received after completion of a sendto() or sendmsg() operation will cause a subsequent sendto() or sendmsg() operation using the same destination address (to parameter) to fail with the appropriate error. See send(3SOCKET).

If the SO_TIMESTAMP option is enabled on a SO_DGRAM or a SO_RAW socket, the recvmsg(3XNET) call will return a timestamp in the native data format, corresponding to when the datagram was received.

The SO_EXCLBIND option is used to enable or disable the exclusive binding of a socket. It overrides the use of the SO_REUSEADDR option to reuse an address on bind(3SOCKET). The actual semantics of the SO_EXCLBIND option depend on the underlying protocol. See tcp(7P) or udp(7P) for more information.

The SO_TYPE and SO_ERROR options are used only with getsockopt(). The SO_TYPE option returns the type of the socket, for example, SOCK_STREAM. It is useful for servers that inherit sockets on startup. The SO_ERROR option returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

The SO_MAC_EXEMPT option is used to toggle socket behavior with unlabeled peers. A socket that has this option enabled can communicate with an unlabeled peer if it is in the global zone or has a label that dominates the default label of the peer. Otherwise, the socket must have a label that is equal to the default label of the unlabeled peer. Calling setsockopt() with this option returns an EACCES error if the process lacks the NET_MAC_AWARE privilege or if the socket is bound. The SO_MAC_EXEMPT option is available only when the system is configured with Trusted Extensions.

The SO_ALLZONES option can be used to bypass zone boundaries between shared-IP zones. Normally, the system prevents a socket from being bound to an address that is not assigned to the current zone. It also prevents a socket that is bound to a wildcard address from receiving traffic for other zones. However, some daemons which run in the global zone might need to send and receive traffic using addresses that belong to other shared-IP zones. If set before a socket is bound, SO_ALLZONES causes the socket to ignore zone boundaries between shared-IP zones and permits the socket to be bound to any address assigned to the shared-IP zones. If the socket is bound to a wildcard address, it receives traffic intended for all shared-IP zones and behaves as if an equivalent socket were bound in each active shared-IP zone. Applications that use the SO_ALLZONES option to initiate connections or send datagram traffic should specify the source address for outbound traffic by binding to a specific

address. There is no effect from setting this option in an exclusive-IP zone. Setting this option requires the sys_net_config privilege. See zones(5).

RETURN VALUES

If successful, getsockopt() and setsockopt() return 0. Otherwise, the functions return -1 and set errno to indicate the error.

ERRORS

The getsockopt() and setsockopt() calls succeed unless:

EBADF	The argument s is not a valid file descriptor.
ENOMEM	There was insufficient memory available for the operation to complete.
ENOPROTOOPT	The option is unknown at the level indicated.
ENOSR	There were insufficient STREAMS resources available for the operation to complete.
ENOTSOCK	The argument s is not a socket.
ENOBUFS	SO_SNDBUF or SO_RCVBUF exceeds a system limit.
EINVAL	Invalid length for IP_OPTIONS.
EHOSTUNREACH	Invalid address for IP_MULTICAST_IF.
EINVAL	Not a multicast address for IP_ADD_MEMBERSHIP and IP_DROP_MEMBERSHIP.
EADDRNOTAVAIL	Bad interface address for IP_ADD_MEMBERSHIP and IP_DROP_MEMBERSHIP.
EADDRINUSE	Address already joined for IP_ADD_MEMBERSHIP.

```
386      ENOENT           Address not joined for IP_DROP_MEMBERSHIP.
```



```
404      EPERM            No permissions.

407      EACCES           Permission denied.

410      EINVAL           The specified  option  is  invalid  at  the
411                       specified  socket  level, or the socket has
412                       been shut down.

415 ATTRIBUTES
416      See attributes(5) for descriptions of the  following  attri-
417      butes:
```

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

```
428 SEE ALSO
429      ndd(1M),   close(2),   ioctl(2),   read(2),   bind(3SOCKET),
430      getprotobyname(3SOCKET),    recv(3SOCKET),   recvmsg(3XNET),
431      send(3SOCKET),  socket(3SOCKET),   socket.h(3HEAD),   attri-
432      butes(5), zones(5), tcp(7P), udp(7P)

435      Solaris Tunable Parameters Reference Manual
```

```
**********************************************************
    4169 Fri Dec 19 04:18:48 2008
new/./if_mpadm.txt
Clearview IPMP manpages
**********************************************************
```

```
System Administration Commands                    if_mpadm(1M)



NAME
     if_mpadm - administer interfaces in an IP multipathing group
     if_mpadm - change operational status of interfaces within  a
     multipathing group

SYNOPSIS
     if_mpadm -d | -r ifname
     /usr/sbin/if_mpadm -d interface_name


     /usr/sbin/if_mpadm -r interface_name


DESCRIPTION
     The *if_mpadm*   utility administers IP  interfaces that are
     part   of  an  IP multipathing   (IPMP)  group.   Currently,
     administration  is limited  to  offlining  IP interfaces and
     undoing previous offline operations.
     Use the if_mpadm utility to change the operational status of
     interfaces that are part of an IP multipathing group. If the
     interface is operational, you can use if_mpadm -d to  detach
     or  off-line  the  interface. If the interface is off-lined,
     use if_mpadm -r to revert it to its original state.

     When  an IP interface is  taken offline, all IP data traffic
     that was flowing over the  IP interface is moved to  another
     IP interface  in the IPMP group.  In  addition,  all *UP* IP
     addresses  hosted  on the   IP interface  are  brought down,
     causing    *in.mpathd(1M)*  to  stop   probe-based   failure
     detection on  the IP interface.   As a result, an offline IP
     interface will  not be used  for any inbound or  outbound IP
     traffic.  Only IP interfaces that  are in an IPMP group  may
     be brought   offline.  If  the  IP  interface is   the  last
     functioning  interface    in  the  IPMP  group, the  offline
     operation will fail.

     When an offline operation is undone, any IP addresses hosted
     on that IP interface are brought *UP* and will be considered
     by *in.mpathd*   for probe-based   failure   detection.   In
     addition, provided the IP interface is otherwise active (see
     *in.mpathd(1M)*), it will again  be used to send and receive
     IP  data  traffic for the  IPMP   group.  Note that  not all
     offline operations can be undone.  For instance, *in.mpathd*
     may have  offlined  an  IP interface  because  its  hardware
     address was not    unique   within its   IPMP  group.    The
     *ipmpstat*  utility  can  be used to    determine why an  IP
     interface is offline,    identify which IP interfaces  in  a
     group are  being used for  inbound and  outbound IP traffic,
     and more; see *ipmpstat(1M)*.
     When a network interface is off-lined,  all  network  access
     fails  over  to a different interface in the IP multipathing
     group. Any addresses that do not failover are brought  down.
     Network  access  includes  unicast, broadcast, and multicast
     for IPv4 and unicast  and  multicast  for  IPv6.   Addresses
     marked  with IFF_NOFAILOVER do not failover. They are marked
     down. After an interface is off-lined, the system  will  not
     use  the  interface for any outbound or inbound traffic, and
```



```
     the interface can be safely removed from the system  without
     any loss of network access.


     The if_mpadm utility can be applied only to interfaces  that
     are part of an IP multipathing group.

OPTIONS
     The *if_mpadm* utility supports the following options:
     The if_mpadm utility supports the following options:

     -d ifname            Offline the  IP interface  specified by
                          *ifname*.   If  *ifname* is  not in  an
                          IPMP group,  or the offline would cause
                          the   IPMP   group    to  lose  network
                          connectivity, the operation will fail.
     -d interface_name    Detach or off-line the interface speci-
                          fied by     interface_name.

     -r ifname            Undo  a  previous  offline   of the  IP
                          interface   specified by *ifname*.   If
                          *ifname* is  not offline, the operation
                          will fail.

     -r interface_name     Reattach or undo the  previous  detach
                          or  off-line operation on the interface
                          specified by interface_name. Unless the
                          -d  option  was  used to detach or off-
                          line the interface,  this  option  will
                          fail.


EXAMPLES
     Example 1 Offlining an IP Interface
     Example 1 Detaching an Interface

     The  following   command  offlines IP  interface   *under0*,
     causing any  IP packets  that  were being sent  and received
     through  it  to be handled  by  another IP  interface in its
     group.

       example% if_mpadm -d under0


     Example 2 Undoing a Previous Offline Operation
SunOS 5.11           Last change: 3 Sep 2002                  1


     Use the following  command to  undo the previous  operation:


       example% if_mpadm -r under0


System Administration Commands                    if_mpadm(1M)


     Use the following command to off-line or detach  the  inter-
     face.  All  network  access will failover from hme0 to other
     interfaces in the same IP multipathing group.  If  no  other
     interfaces are in the same group, the operation will fail.


       example% if_mpadm -d hme0
```

Example 2 Reattaching an Off-line Interface

Use the following command to undo the previous operation. Network access will failback to hme0.

```
example% if_mpdadm -r hme0
```

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Unstable

SEE ALSO

ifconfig(1M), in.mpathd(1M), attributes(5)

DIAGNOSTICS

cannot offline: no other functioning interfaces are in its IPMP group.

off-line failed as there is no other functional interface available in the multipathing group for failing over the network access.

Description:

This message means that offlining the IP interface would leave the IPMP group without network connectivity.

cannot offline: not a physical interface or not in an IPMP group

This message means that other interfaces in the group are failed over already or the multipathing configuration was not suitable for completing a failover.

off-line cannot be undone because multipathing configuration is not consistent across all the interfaces in the group.

Description:

This means that the IP interface is not an underlying interface in an IPMP group, and therefore is not eligible to be offlined.

This message means that some interfaces in the IP multipathing group are not configured consistently with other interfaces in the group, for example, one of the interfaces in the group does not have an IFF_NOFAILOVER address.

****
5836 Fri Dec 19 04:18:50 2008
new/./if_nameindex.txt
Clearview IPMP manpages
****

NAME

if_nametoindex, if_indextoname, if_nameindex, if_freenameindex - routines to map Internet Protocol network interface names and interface indexes

SYNOPSIS

```
cc [ flag... ] file... -lsocket [ library... ]
#include <net/if.h>

unsigned int if_nametoindex(const char *ifname);

char *if_indextoname(unsigned int ifindex, char *ifname);

struct if_nameindex *if_nameindex(void)

void if_freenameindex(struct if_nameindex *ptr);
```

PARAMETERS

ifname	interface name
ifindex	interface index
ptr	pointer returned by if_nameindex()

DESCRIPTION

This API defines two functions that map between an Internet Protocol network interface name and index, a third function that returns all the interface names and indexes, and a fourth function to return the dynamic memory allocated by the previous function.

Network interfaces are normally known by names such as eri0, sl1, ppp2, and the like. The ifname argument must point to a buffer of at least IF_NAMESIZE bytes into which the interface name corresponding to the specified index is returned. IF_NAMESIZE is defined in <net/if.h> and its value includes a terminating null byte at the end of the interface name.

if_nametoindex() The if_nametoindex() function returns the interface index corresponding to the interface name pointed to by the ifname pointer. If the specified interface name does not exist, the return value is 0, and errno is set to

ENXIO. If there was a system error, such as running out of memory, the return value is 0 and errno is set to the proper value, for example, ENOMEM.

if_indextoname() The if_indextoname() function maps an interface index into its corresponding name. This pointer is also the return value of the function. If there is no interface corresponding to the specified index, NULL is returned, and errno is set to ENXIO, if there was a system error, such as running out of memory, if_indextoname() returns NULL and errno would be set to the proper value, for example, ENOMEM.

if_nameindex() **The if_nameindex() function returns an**
**if_nameindex()* *The if_nameindex() function returns an* array of if_nameindex structures, one structure per interface. The if_nameindex structure holds the information about a single interface and is defined when the <net/if.h> header is included:

```
struct if_nameindex
  unsigned int   if_index; /* 1, 2, ... */
  char          *if_name;  /* "net0", ... */
  char          *if_name;  /* null terminated name:
};
```

While any IPMP IP interfaces are returned by *if_nameindex()*, the underlying IP interfaces that comprise each IPMP group are not returned.

The end of the array of structures is indicated by a structure with an if_index of 0 and an if_name of NULL. The function returns a null pointer upon an error and sets errno to the appropriate value. The memory used for this array of structures along with the interface names pointed to by the if_name members is obtained dynamically. This memory is freed by the if_freenameindex() function.

if_freenameindex() The if_freenameindex() function frees the dynamic memory that was allocated by if_nameindex(). The argument to this function must be a pointer that was returned by if_nameindex().

142 ATTRIBUTES
143 See attributes(5) for descriptions of the following attri-
144 butes:

148
149 | ATTRIBUTE TYPE | ATTRIBUTE VALUE |
150 |---|---|
151 | Availability | SUNWcsl (32-bit) |
152 | | SUNWcslx (64-bit) |
153 |---|---|
154 | MT Level | MT Safe |
155 |---|---|

158 SEE ALSO
159 ifconfig(1M), if_nametoindex(3XNET), attributes(5), if(7P)

```
**********************************************************
   27477 Fri Dec 19 04:18:51 2008
new/./if_tcp.txt
Clearview IPMP manpages
**********************************************************
Protocols                                                if_tcp(7P)

NAME
     if_tcp, if - general properties of Internet Protocol network
     interfaces

DESCRIPTION
     A network interface is a device for  sending  and  receiving
     packets  on  a  network.  It  is  usually a hardware device,
     although it can be implemented in software.  Network  inter-
     faces  used  by the Internet Protocol (IPv4 or IPv6) must be
     STREAMS devices conforming to the Data Link Provider  Inter-
     face (DLPI). See dlpi(7P).

APPLICATION PROGRAMMING INTERFACE
     An interface becomes available to IP when it is  opened  and
     the  IP  module  is  pushed  onto the stream with the I_PUSH
     ioctl(2)  command.  (See  streamio(7I)).  The   SIOCSLIFNAME
     ioctl(2)  is issued to specify the name of the interface and
     to indicate whether it is IPv4 or IPv6.  This  may  be  ini-
     tiated by the kernel at boot time or by a user program after
     the system is running. Each interface must be assigned an IP
     address with the SIOCSLIFADDR ioctl() before it can be used.
     On interfaces where the network-to-link layer  address  map-
     ping  is  static,  only the network number is taken from the
     ioctl() request;  the  remainder  is  found  in  a  hardware
     specific   manner.   On  interfaces  which  provide  dynamic
     network-to-link layer address mapping facilities (for  exam-
     ple,  10Mb/s  Ethernets  using  arp(7P)), the entire address
     specified in the ioctl() is used. A routing table entry  for
     destinations  on  the  network of the interface is installed
     automatically when an interface's address is set.

     Note   IPMP  IP  interfaces cannot   be  created  using  the
     procedure   described above.  Instead,  ifconfig(1M) must be
     used to create IPMP IP interfaces.

IOCTLS
     The following ioctl() calls may be  used  to  manipulate  IP
     network  interfaces. Unless specified otherwise, the request
     takes an lifreq structure as its parameter.  This  structure
     has the form:

       struct lifreq {
       #define LIFNAMSIZ   32
         char    lifr_name[LIFNAMSIZ];       /* if name, e.g. "le1" */
         union {
            int    lifru_addrlen;            /* for subnet/token etc */
            uint_t lifru_ppa;                /* SIOCSLIFNAME */
         } lifr_lifru1;
         union {
            struct sockaddr_storage lifru_addr;
            struct sockaddr_storage lifru_dstaddr;
            struct sockaddr_storage lifru_broadaddr;
            struct sockaddr_storage lifru_token;    /* With lifr_addrlen */
            struct sockaddr_storage lifru_subnet;   /* With lifr_addrlen */
            int                     lifru_index;    /* interface index */
            uint64_t                lifru_flags;    /* SIOC?LIFFLAGS */

SunOS 5.10          Last change: 14 Nov 2008                  1
```



```
Protocols                                                if_tcp(7P)

            int                     lifru_metric;
            uint_t                  lifru_mtu;
            int                     lif_muxid[2];   /* mux id's for arp & ip */
            struct lif_nd_req       lifru_nd_req;
            struct lif_ifinfo_req   lifru_ifinfo_req;
            zoneid_t                lifru_zone;     /* SIOC[GS]LIFZONE */
         } lifr_lifru;

       #define lifr_addrlen   lifr_lifru1.lifru_addrlen
       #define lifr_ppa       lifr_lifru1.lifru_ppa       /* Driver's ppa */
       #define lifr_addr      lifr_lifru.lifru_addr       /* address */
       #define lifr_dstaddr   lifr_lifru.lifru_dstaddr
       #define lifr_broadaddr lifr_lifru.lifru_broadaddr  /* broadcast addr. */
       #define lifr_token     lifr_lifru.lifru_token      /* address token */
       #define lifr_subnet    lifr_lifru.lifru_subnet     /* subnet prefix */
       #define lifr_index     lifr_lifru.lifru_index      /* interface index */
       #define lifr_flags     lifr_lifru.lifru_flags      /* flags */
       #define lifr_metric    lifr_lifru.lifru_metric     /* metric */
       #define lifr_mtu       lifr_lifru.lifru_mtu        /* mtu */
       #define lifr_ip_muxid  lifr_lifru.lif_muxid[0]
       #define lifr_arp_muxid lifr_lifru.lif_muxid[1]
       #define lifr_nd        lifr_lifru.lifru_nd_req     /* SIOCLIF*ND */
       #define lifr_ifinfo    lifr_lifru.lifru_ifinfo_req /* SIOC[GS]LIFLNKINFO
       #define lifr_zone      lifr_lifru.lifru_zone       /* SIOC[GS]LIFZONE */
       };

     SIOCSLIFADDR        Set interface address.

     SIOCGLIFADDR        Get interface address.

     SIOCSLIFDSTADDR     Set point to point  address  for  inter-
                         face.

     SIOCGLIFDSTADDR     Get point to point  address  for  inter-
                         face.

     SIOCSLIFFLAGS       Set interface flags field. If the inter-
                         face   is  marked  down,  any  processes
                         currently routing  packets  through  the
                         interface are notified.

     SIOCGLIFFLAGS       Get interface flags.

     SIOCGLIFCONF        Get  interface configuration list.  This
                         request takes a *lifconf* structure (see
                         below) as a value-result parameter.  The
                         *lifc_family*   field can  be  set    to
                         *AF_UNSPEC* to retrieve  both  *AF_INET*
                         and    *AF_INET6*    interfaces.     The
                         *lifc_len* field  should  be set  to the
                         size  of  the  buffer    pointed to   by
                         *lifc_buf*.
                         request  takes an lifconf structure (see

                         The *lifc_flags* field should usually be
                         set to zero, but  callers that need low-
                         level knowledge    of the  underlying IP
                         interfaces that comprise an IPMP   group
                         can   set  it  to  *LIFC_UNDER_IPMP*  to
                         request that   those     interfaces   be
                         included in the result.
SunOS 5.10          Last change: 14 Nov 2008                  2

                         Upon success,   *lifc_len*  will contain
```

the length, in bytes, of the array of *lifreq* structures pointed to by *lifc_req*. For each *lifreq* structure, the *lifr_name* and *lifr_addr* fields will be valid.

below) as a value-result parameter. The lifc_family field can be set to AF_UNSPEC to retrieve both AF_INET and AF_INET6 interfaces. The lifc_flags field should be set to zero. The lifc_len field should be set to the size of the buffer pointed to by lifc_buf. Upon success, lifc_len will contain the length, in bytes, of the array of lifreq structures pointed to by lifc_req. For each lifreq structure, the lifr_name and lifr_addr fields will be valid.

SIOCGLIFNUM Get number of interfaces. This request returns an integer which is the number of interface descriptions (struct lifreq) that will be returned by the SIOCGLIFCONF ioctl; that is, it gives an **indication of how large lifc_len has to be.**
indication of how large lifc_len has to be. This request takes an lifnum structure (see below) as a value-result parameter. The lifn_family field should be set to AF_UNSPEC to count both AF_INET and AF_INET6 interfaces. The lifn_flags field should be initially set to zero.

This request takes a *struct lifnum* (see below) as a value-result parameter. The *lifn_family* field can be set to *AF_UNSPEC* to count both *AF_INET* and *AF_INET6* interfaces. The *lifn_flags* field should usually be set to zero, but callers that need low-level knowledge of the underlying IP interfaces that comprise an IPMP group can set it to *LIFC_UNDER_IPMP* to request that those interfaces be included in the count.

SIOCSLIFMTU Set the maximum transmission unit (MTU) size for interface. Place the request in the lifru_mtu field. The MTU can not exceed the physical MTU limitation (which is reported in the DLPI DL_INFO_ACK message).

SIOCGLIFMTU Get the maximum transmission unit size for interface.

SIOCSLIFMETRIC Set the metric associated with the interface. The metric is used by routing daemons such as in.routed(1M).

SIOCGLIFMETRIC Get the metric associated with the interface.

SIOCGLIFMUXID Get the ip and arp muxid associated with the interface.

SIOCSLIFMUXID Set the ip and arp muxid associated with the interface.

SIOCGLIFINDEX Get the interface index associated with the interface.

SIOCSLIFINDEX Set the interface index associated with the interface.

SIOCGLIFZONE Get the zone associated with the interface.

SIOCSLIFZONE Set the zone associated with the interface. Only applies for zones that use the shared-IP instance.

SIOCLIFADDIF Add a new logical interface on a physical interface using an unused logical **interface number.**
unit number. If the physical interface is part of an IP multipathing group, the logical interface may be added to a different physical interface in the same group. Upon return, the lifr_name field contains the name of the actual logical interface created.

SIOCLIFREMOVEIF Remove a logical interface by specifying its IP address or logical interface **name.**
name. When the IP address is specified and the interface is part of an IP multipathing group, the logical interface is removed from the physical interface in the group which holds the IP address.

SIOCSLIFTOKEN Set the address token used to form IPv6 link-local addresses and for stateless address autoconfiguration.

SIOCGLIFTOKEN Get the address token used to form IPv6 link-local addresses and for stateless address autoconfiguration.

SIOCSLIFSUBNET Set the subnet prefix associated with the interface.

SIOCGLIFSUBNET Get the subnet prefix associated with the interface.

SIOCSLIFLNKINFO Set link specific parameters for the interface.

SIOCGLIFLNKINFO Get link specific parameters for the interface.

```
226      SIOCLIFDELND        Delete a neighbor cache entry for  IPv6.

228      SIOCLIFGETND        Get a neighbor cache entry for  IPv6.

230      SIOCLIFSETND        Set a neighbor cache entry for  IPv6.

232      SIOCSLIFUSESRC      Set the interface from which to choose a
233                          source address. The lifr_index field has
234                          the interface index corresponding to the
235                          interface whose address is to be used as
236                          the source address for packets going out
237                          on  the interface whose name is provided
238                          by lifr_name. If the lifr_index field is
239                          set  to  zero,  the  previous setting is
240                          cleared.  See ifconfig(1M) for  examples
241                          of the usesrc option.

243      SIOCGLIFUSESRC      Get the interface index of the interface
244                          whose  address  is  used  as  the source
245                          address for packets  going  out  on  the
246                          interface  provided  by lifr_name field.
247                          The value is retrieved in the lifr_index
248                          field.  See ifconfig(1M) for examples of
249                          the usesrc option.

251      SIOCGLIFSRCOF       Get the interface configuration list for
252                          interfaces that use an address hosted on
253                          the   interface    provided    by    the
254                          lifs_ifindex   field   in  the  lifsrcof
255                          struct (see below), as a source address.
256                          The  application sets lifs_maxlen to the

258 SunOS 5.10          Last change: 14 Nov 2008                  5

260 Protocols                                             if_tcp(7P)

262                          size (in bytes) of  the  buffer  it  has
263                          allocated  for  the data. On return, the
264                          kernel sets lifs_len to the actual  size
265                          required.  Note,  the  application could
266                          set lifs_maxlen to  zero  to  query  the
267                          kernel   of  the  required  buffer  size
268                          instead of estimating a buffer size. The
269                          application     tests     lifs_len    <=
270                          lifs_maxlen  --  if  that's  true,   the
271                          buffer  was  big enough and the applica-
272                          tion has an accurate  list.   If  it  is
273                          false,  it  needs  to  allocate a bigger
274                          buffer and try again, and lifs_len  pro-
275                          vides a hint of how big to make the next
276                          trial. See ifconfig(1M) for examples  of
277                          the usesrc option.

279      SIOCTONLINK         Test if the address is  directly  reach-
280                          able,   for  example,  that  it  can  be
281                          reached without going through a  router.
282                          This   request   takes  an  sioc_addrreq
283                          structure (see below) as a  value-result
284                          parameter.  The  sa_addr field should be
285                          set to the address to test.  The  sa_res
286                          field  will  contain a non-zero value if
287                          the address is onlink.

289      SIOCTMYADDR         Test if the address is assigned to  this
290                          node. This request takes an sioc_addrreq
```



```
291                          structure (see below) as a  value-result
292                          parameter.  The  sa_addr field should be
293                          set to the address to test.  The  sa_res
294                          field  will  contain a non-zero value if
295                          the address is assigned to this node.

297      SIOCTMYSITE         Test if the address is part of the  same
298                          site as this node. This request takes an
299                          sioc_addrreq structure (see below) as  a
300                          value-result   parameter.   The  sa_addr
301                          field should be set to  the  address  to
302                          test.  The  sa_res  field will contain a
303                          non-zero value if the address is in  the
304                          same site.

306        The structure used by SIOCGLIFCONF has the form:

308        struct lifconf {
309          sa_family_t     lifc_family;

311 SunOS 5.10          Last change: 14 Nov 2008                  6

313 Protocols                                             if_tcp(7P)

315          int             lifc_flags;       /* request specific
316                                             /* interfaces */
317          int             lifc_len;         /* size of assoc. buffer */
318          union {
319             caddr_t      lifcu_buf;
320             struct lifreq *lifcu_req;
321          } lifc_lifcu;

323        #define lifc_buf lifc_lifcu.lifcu_buf   /* buffer address */
324        #define lifc_req lifc_lifcu.lifcu_req   /* array of structs returned */
325        };

327        The structure used by SIOCGLIFNUM has the form:

329        struct lifnum {
330          sa_family_t     lifn_family;
331          int             lifn_flags;   /* req. specf. interfaces */
332          int             lifn_count;   /* Result */
333        };

335        The structure used by SIOCTONLINK, SIOCTMYADDR and SIOCTMYSITE has the fo

337        struct sioc_addrreq {
338          struct sockaddr_storage sa_addr; /* Address to test */
339          int                     sa_res;  /* Result - 0/1 */
340        };

342        The structure used by SIOCGLIFSRCOF has the form:

344        struct lifsrcof {
345            uint_t  lifs_ifindex;   /* addr on this interface */
346                                    /* used as the src addr */
347            size_t  lifs_maxlen;    /* size of buffer: input */
348            size_t  lifs_len;       /* size of buffer: output */
349            union {
350                   caddr_t lifsu_buf;
351                   struct  lifreq *lifsu_req;
352          } lifs_lifsu;
353        #define lifs_buf lifs_lifsu.lifsu_buf /* buffer addr. */
354        #define lifs_req lifs_lifsu.lifsu_req /* array returned */
355        };
```

The following ioctl() calls are maintained for compatibility but only apply to IPv4 network interfaces, since the data structures are too small to hold an IPv6 address. Unless specified otherwise, the request takes an ifreq structure as its parameter. This structure has the form:

```
  struct ifreq {
```



```
        #define IFNAMSIZ    16
        char ifr_name[IFNAMSIZ];             /* interface name - e.g. "hme0" */
        union {
                struct sockaddr  ifru_addr;
                struct sockaddr  ifru_dstaddr;
                struct sockaddr  ifru_broadaddr;
                short ifru_flags;
                int   ifru_metric;
                int   if_muxid[2];                  /* mux id's for arp and ip *
                int   ifru_index;                     /* interface index */
             } ifr_ifru;

        #define ifr_addr      ifr_ifru.ifru_addr       /* address */
        #define ifr_dstaddr   ifr_ifru.ifru_dstaddr    /* other end of p-to-p lin
        #define ifr_broadaddr ifr_ifru.ifru_broadaddr  /* broadcast address */
        #define ifr_flags     ifr_ifru.ifru_flags      /* flags */
        #define ifr_index     ifr_ifru.ifru_index      /* interface index  */
        #define ifr_metric    ifr_ifru.ifru_metric     /* metric */
        };
```

SIOCSIFADDR — Set interface address.

SIOCGIFADDR — Get interface address.

SIOCSIFDSTADDR — Set point to point address for interface.

SIOCGIFDSTADDR — Get point to point address for interface.

SIOCSIFFLAGS — Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.

SIOCGIFFLAGS — Get interface flags.

SIOCGIFCONF — Get interface configuration list. This request takes an ifconf structure (see below) as a value-result parameter. The ifc_len field should be set to the size of the buffer pointed to by ifc_buf. Upon success, ifc_len will contain the length, in bytes, of the array of ifreq structures pointed to by ifc_req. For each ifreq structure, the ifr_name and ***ifr_addr* fields will be valid. While any IPMP IP interfaces will be included in the array, the underlying IP interfaces that comprise those IPMP groups will not be.**
ifr_addr fields will be valid.

SIOCGIFNUM — Get number of interfaces. This request returns an integer which is the number of interface descriptions (struct ifreq) that will be returned by the SIOCGIFCONF ioctl; that is, it gives an indication of **how large *ifc_len* has to be. While any IPMP IP interfaces will be included in the array, the underlying IP interfaces that comprise those IPMP groups will not be.**
how large ifc_len has to be.

SIOCSIFMTU — Set the maximum transmission unit (MTU) size for interface. Place the request in the ifr_metric field. The MTU has to be smaller than physical MTU limitation (which is reported in the DLPI DL_INFO_ACK message).

SIOCGIFMTU — Get the maximum transmission unit size for interface. Upon success, the request is placed in the ifr_metric field.

SIOCSIFMETRIC — Set the metric associated with the interface. The metric is used by routine daemons such as in.routed(1M).

SIOCGIFMETRIC — Get the metric associated with the interface.

SIOCGIFMUXID — Get the ip and arp muxid associated with the interface.

SIOCSIFMUXID — Set the ip and arp muxid associated with the interface.

SIOCGIFINDEX — Get the interface index associated with the interface.

SIOCSIFINDEX — Set the interface index associated with the interface.

The ifconf structure has the form:

```
  struct ifconf {
  int  ifc_len;                        /* size of assoc. buffer */
  union {
```



```
               caddr_t       ifcu_buf;
               struct ifreq  *ifcu_req;
            } ifc_ifcu;

        #define ifc_buf   ifc_ifcu.ifcu_buf   /* buffer address */
        #define ifc_req   ifc_ifcu.ifcu_req   /* array of structs returned */
        };
```

IFF_ Flags

You can use the ifconfig(1M) command to display the IFF_ flags listed below (with the leading IFF_ prefix removed). See the ifconfig(1M) manpage for a definition of each flag.

```
487         #define IFF_UP          0x0000000001    /* Address is up */
471         #define IFF_UP          0x0000000001    /* Interface is up */
488         #define IFF_BROADCAST   0x0000000002    /* Broadcast address valid */
489         #define IFF_DEBUG       0x0000000004    /* Turn on debugging */
490         #define IFF_LOOPBACK    0x0000000008    /* Loopback net */

492         #define IFF_POINTOPOINT 0x0000000010    /* Interface is p-to-p */
493         #define IFF_NOTRAILERS  0x0000000020    /* Avoid use of trailers */
494         #define IFF_RUNNING     0x0000000040    /* Resources allocated */
495         #define IFF_NOARP       0x0000000080    /* No address res. protocol */

497         #define IFF_PROMISC     0x0000000100    /* Receive all packets */
498         #define IFF_ALLMULTI    0x0000000200    /* Receive all multicast pkts */
499         #define IFF_INTELLIGENT 0x0000000400    /* Protocol code on board */
500         #define IFF_MULTICAST   0x0000000800    /* Supports multicast */

502         #define IFF_MULTI_BCAST 0x0000001000    /* Multicast using broadcst. add.
503         #define IFF_UNNUMBERED  0x0000002000    /* Non-unique address */
504         #define IFF_DHCPRUNNING 0x0000004000    /* DHCP controls interface */
505         #define IFF_PRIVATE     0x0000008000    /* Do not advertise */

507         #define IFF_NOXMIT      0x0000010000    /* Do not transmit pkts */
508         #define IFF_NOLOCAL     0x0000020000    /* No address - just on-link subn
509         #define IFF_DEPRECATED  0x0000040000    /* Address is deprecated */
493         #define IFF_DEPRECATED  0x0000040000    /* Interface addr. deprecated */
510         #define IFF_ADDRCONF    0x0000080000    /* Addr. from stateless addrconf

512         #define IFF_ROUTER      0x0000100000    /* Router on interface */
513         #define IFF_NONUD       0x0000200000    /* No NUD on interface */
514         #define IFF_ANYCAST     0x0000400000    /* Anycast address */
515         #define IFF_NORTEXCH    0x0000800000    /* Don't xchange rout. info */

517         #define IFF_IPV4        0x0001000000    /* IPv4 interface */
518         #define IFF_IPV6        0x0002000000    /* IPv6 interface */
519         #define IFF_NOFAILOVER  0x0008000000    /* in.mpathd test address */
520         #define IFF_FAILED      0x0010000000    /* Interface has failed */
503         #define IFF_NOFAILOVER  0x0008000000    /* No failover on NIC fail. */
504         #define IFF_FAILED      0x0010000000    /* NIC has failed */

522         #define IFF_STANDBY     0x0020000000    /* Interface is a hot-spare */
523         #define IFF_INACTIVE    0x0040000000    /* Functioning but not used */
524         #define IFF_OFFLINE     0x0080000000    /* Interface is offline */
506         #define IFF_STANDBY     0x0020000000    /* Standby NIC-use on fail. */
507         #define IFF_INACTIVE    0x0040000000    /* Stndby active or not? */
508         #define IFF_OFFLINE     0x0080000000    /* NIC offlined */

510 SunOS 5.10          Last change: 14 Nov 2008                  10

512 Protocols                                               if_tcp(7P)

525         #define IFF_XRESOLV     0x0100000000    /* IPv6 external resolver */

527         #define IFF_COS_ENABLED 0x0200000000    /* If CoS marking is supported */
528         #define IFF_PREFERRED   0x0400000000    /* Prefer as source address */
529         #define IFF_TEMPORARY   0x0800000000    /* RFC3041 */
530         #define IFF_FIXEDMTU    0x1000000000    /* MTU set with SIOCSLIFMTU */

532         #define IFF_VIRTUAL     0x2000000000    /* Cannot send/receive pkts */
533         #define IFF_DUPLICATE   0x4000000000    /* Local address in use */
534         #define IFF_IPMP        0x8000000000    /* IPMP IP interface */

536 ERRORS
537      EPERM        Calling process has insufficient privileges.

539      ENXIO        The  lifr_name member of the  lifreq  structure
540                   contains an invalid value.
```



```
542                   For SIOCGLIFSRCOF, the lifs_ifindex  member  of
543                   the  lifsrcof  structure  contains  an  invalid
544                   value.

546                   For SIOCSLIFUSESRC, this error is  returned  if
547                   the lifr_index is set to an invalid value.

549      EBADADDR     Wrong address family or malformed address.

551      EBUSY        For  SIOCSLIFFLAGS, this error is returned when
552                   the  order  of bringing  the physical interface
553                   (for example,  eri0) and  a  logical  interface
554                   associated  with  the  same  physical interface
555                   (for example, eri0:1) up or down  is  violated.
556                   The  physical  interface  must be configured up
557                   first  and cannot be configured down until  all
558                   the corresponding  logical interfaces have been
559                   configured down.

561      EINVAL       For SIOCSLIFMTU, this error  is  returned  when
562                   the  requested  MTU size is invalid. This error
563                   indicates the MTU size is greater than the  MTU
564                   size  supported  by  the  DLPI provider or less
565                   than 68 (for  IPv4)  or  less  than  1280  (for
566                   IPv6).

568                   For SIOCSLIFUSESRC, this error is  returned  if
569                   either  the  lifr_index  or  lifr_name identify
570                   interfaces that are already part of an existing
571                   IPMP group.

573 SunOS 5.10          Last change: 14 Nov 2008                  11

575 Protocols                                               if_tcp(7P)

577      EEXIST       For SIOCLIFADDIF, this error is returned if the
578                   lifr_name   member   in  the  lifreq  structure
579                   corresponds to an interface  that  already  has
580                   the PPA specified by lifr_ppa plumbed.

582 SEE ALSO
583      ifconfig(1M),   in.routed(1M),    ioctl(2),    streamio(7I),
584      arp(7P), dlpi(7P), ip(7P), ip6(7P)
```

****
71559 Fri Dec 19 04:18:53 2008
new/./ifconfig.txt
Clearview IPMP manpages
****

```
  1 System Administration Commands                        ifconfig(1M)



  5 NAME
  6      ifconfig - configure network interface parameters

  8 SYNOPSIS
  9      ifconfig interface [address_family] [address  [/prefix_length]
 10      [dest_address]] [addif  address  [/prefix_length]]
 11      [removeif  address  [/prefix_length]] [arp |  -arp]
 12      [auth_algs authentication algorithm] [encr_algs encryption algorithm]
 13      [encr_auth_algs authentication algorithm] [auto-revarp]
 14      [broadcast  address] [deprecated |  -deprecated]
 15      [preferred |  -preferred] [destination  dest_address]
 16      [ether  [address]] [failover |  -failover] [group
 17      [name |  ""]] [index if_index] [ipmp] [metric  n] [modlist]
 20      [name |  ""]] [index   if_index] [metric  n] [modlist]
 18      [modinsert mod_name@pos] [modremove mod_name@pos]
 19      [mtu  n] [netmask  mask] [plumb] [unplumb] [private
 20      |  -private] [nud |  -nud] [set  [address]  [/netmask]]
 21      [standby |  -standby] [subnet  subnet_address] [tdst
 22      tunnel_dest_address] [token   address/prefix_length]
 23      [tsrc  tunnel_src_address] [trailers |  -trailers]
 24      [up] [down] [usesrc [name |  none]] [xmit |  -xmit]
 25      [encaplimit n |  -encaplimit] [thoplimit n] [router
 26      |  -router] [zone zonename |  -zone |  -all-zones]


 29      ifconfig [address_family] interface {auto-dhcp |  dhcp} [primary]
 30      [wait  seconds]  drop |  extend |  inform |  ping
 31      |  release |  start |  status


 34 DESCRIPTION
 35      The command ifconfig is used to assign an address to a  net-
 36      work  interface  and  to configure network interface parame-
 37      ters. The ifconfig command must be  used  at  boot  time  to
 38      define  the  network  address of each interface present on a
 39      machine; it may also be used at a later time to redefine  an
 40      interface's  address  or  other  operating parameters. If no
 41      option is specified, ifconfig displays  the  current  confi-
 42      guration  for  a  network interface. If an address family is
 43      specified, ifconfig reports only  the  details  specific  to
 44      that  address  family.  Only privileged users may modify the
 45      configuration of  a  network  interface.  Options  appearing
 46      within  braces ({}) indicate that one of the options must be
 47      specified.

 49   DHCP Configuration
 50      The forms of ifconfig that use the auto-dhcp or  dhcp  argu-
 51      ments  are  used  to  control the Dynamic Host Configuration
 52      Protocol ("DHCP") configuration of the  interface.  In  this
 53      mode,   ifconfig   is   used   to   control   operation   of
 54      dhcpagent(1M), the DHCP client daemon. Once an interface  is
 55      placed under DHCP control by using the start operand, ifcon-
 56      fig should not, in normal operation, be used to  modify  the
```



```
 60 SunOS 5.11          Last change: 21 Jan 2007                  1




 67 System Administration Commands                        ifconfig(1M)


 71      address  or characteristics of the interface. If the address
 72      of an interface under DHCP is changed, dhcpagent will remove
 73      the interface from its control.

 75 OPTIONS
 76      The following options are supported:

 78      addif address

 80          Create the next unused logical interface on  the  speci-
 81          fied  physical  interface.
 84          fied  physical  interface.  If the physical interface is
 85          part of a multipathing group, the logical interface  can
 86          be  added  to a different physical interface in the same
 87          group.


 83      all-zones

 85          Make the interface available to every shared-IP zone  on
 86          the  system.  The  appropriate  zone to which to deliver
 87          data is determined using the  tnzonecfg  database.  This
 88          option  is  available  only  if the system is configured
 89          with the Solaris Trusted Extensions feature.

 91          The tnzonecfg database is described in the  tnzonecfg(4)
 92          man  page,  which  is part of the Solaris Trusted Exten-
 93          sions Reference Manual.


 96      anycast

 98          Marks the logical interface as  an  anycast  address  by
 99          setting  the ANYCAST flag. See "INTERFACE FLAGS," below,
100          for more information on anycast.


103      -anycast

105          Marks the logical interface as not an anycast address by
106          clearing the ANYCAST flag.


109      arp

111          Enable  the  use  of  the  Address  Resolution  Protocol
112          ("ARP")  in  mapping between network level addresses and
113          link level addresses (default). This is currently imple-
114          mented  for  mapping  between  IPv4  addresses  and  MAC
115          addresses.
```

-arp

Disable the use of the ARP on a physical interface.
ARP cannot be disabled on an IPMP IP interface.

auth_algs authentication algorithm

For a tunnel, enable IPsec AH with the authentication algorithm specified. The algorithm can be either a number or an algorithm name, including any to express no preference in algorithm. All IPsec tunnel properties must be specified on the same command line. To disable tunnel security, specify an auth_alg of none.

It is now preferable to use the ipsecconf(1M) command when configuring a tunnel's security properties. If ipsecconf was used to set a tunnel's security properties, this keyword will not affect the tunnel.

auto-dhcp

Use DHCP to automatically acquire an address for this interface. This option has a completely equivalent alias called dhcp.

For IPv6, the interface specified must be the zeroth logical interface (the physical interface name), which has the link-local address.

primary

Defines the interface as the primary. The interface is defined as the preferred one for the delivery of client-wide configuration data. Only one interface can be the primary at any given time. If another interface is subsequently selected as the primary, it replaces the previous one. Nominating an interface as the primary one will not have much significance once the client work station has booted, as many applications will already have started and been configured with data read from the previous primary interface.

wait seconds

The ifconfig command will wait until the operation either completes or for the interval specified, whichever is the sooner. If no wait interval is given, and the operation is one that cannot complete

immediately, ifconfig will wait 30 seconds for the

requested operation to complete. The symbolic value forever may be used as well, with obvious meaning.

drop

Remove the specified interface from DHCP control without notifying the DHCP server, and record the current lease for later use. Additionally, for IPv4, set the IP address to zero and mark the interface as "down." For IPv6, unplumb all logical interfaces plumbed by dhcpagent.

extend

Attempt to extend the lease on the interface's IP address. This is not required, as the agent will automatically extend the lease well before it expires.

inform

Obtain network configuration parameters from DHCP without obtaining a lease on IP addresses. This is useful in situations where an IP address is obtained through mechanisms other than DHCP.

ping

Check whether the interface given is under DHCP control, which means that the interface is managed by the DHCP agent and is working properly. An exit status of 0 means success.

release

Relinquish the IP addresses on the interface by notifying the server and discard the current lease. For IPv4, mark the interface as "down." For IPv6, all logical interfaces plumbed by dhcpagent are unplumbed.

start

Start DHCP on the interface.

status

Display the DHCP configuration status of the inter-face.

auto-revarp

Use the Reverse Address Resolution Protocol (RARP) to automatically acquire an address for this interface. This will fail if the interface does not support RARP; for example, IPoIB (IP over InfiniBand), and on IPv6 interfaces.

broadcast address

For IPv4 only. Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 1's. A "+" (plus sign) given for the broadcast value causes the broadcast address to be reset to a default appropriate for the (possibly new) address and netmask. The arguments of ifconfig are interpreted left to right. Therefore

```
example% ifconfig -a netmask + broadcast +
```

and

```
example% ifconfig -a broadcast + netmask +
```

may result in different values being assigned for the broadcast addresses of the interfaces.

deprecated

Marks the logical interface as deprecated. An address associated with a deprecated interface will not be used as source address for outbound packets unless either there are no other addresses available on the interface or the application has bound to this address explicitly. The status display shows DEPRECATED as part of flags. See for information on the flags supported by ifconfig.

-deprecated

Marks a logical interface as not deprecated. An address

associated with such an interface could be used as a source address for outbound packets.

preferred

Marks the logical interface as preferred. This option is only valid for IPv6 addresses. Addresses assigned to preferred logical interfaces are preferred as source addresses over all other addresses configured on the system, unless the address is of an inappropriate scope relative to the destination address. Preferred addresses are used as source addresses regardless of which physi-cal interface they are assigned to. For example, you can configure a preferred source address on the loopback interface and advertise reachability of this address by using a routing protocol.

-preferred

Marks the logical interface as not preferred.

destination dest_address

Set the destination address for a point-to point inter-face.

dhcp

This option is an alias for option auto-dhcp

down

Mark a logical interface as "down". (That is, turn off the IFF_UP bit.) When a logical interface is marked "down," the system does not attempt to use the address assigned to that interface as a source address for out-bound packets and will not recognize inbound packets destined to that address as being addressed to this host. Additionally, when all logical interfaces on a given physical interface are "down," the physical inter-face itself is disabled.

When a logical interface is down, all routes that specify that interface as the output (using the -ifp option in the route(1M) command or RTA_IFP in a route(7P) socket) are removed from the forwarding table.

Routes marked with RTF_STATIC are returned to the table

if the interface is brought back up, while routes not marked with RTF_STATIC are simply deleted.

When all logical interfaces that could possibly be used to reach a particular gateway address are brought down (specified without the interface option as in the previous paragraph), the affected gateway routes are treated as though they had the RTF_BLACKHOLE flag set. All matching packets are discarded because the gateway is unreachable.

encaplimit n

Set the tunnel encapsulation limit for the interface to n. This option applies to IPv4-in-IPv6 and IPv6-in-IPv6 tunnels only. The tunnel encapsulation limit controls how many more tunnels a packet may enter before it leaves any tunnels, that is, the tunnel nesting level.

-encaplimit

Disable generation of the tunnel encapsulation limit. This option applies only to IPv4-in-IPv6 and IPv6-in-IPv6 tunnels.

encr_auth_algs authentication algorithm

For a tunnel, enable IPsec ESP with the authentication algorithm specified. It can be either a number or an algorithm name, including any or none, to indicate no algorithm preference. If an ESP encryption algorithm is specified but the authentication algorithm is not, the default value for the ESP authentication algorithm will be any.

It is now preferable to use the ipsecconf(1M) command when configuring a tunnel's security properties. If ipsecconf was used to set a tunnel's security properties, this keyword will not affect the tunnel.

encr_algs encryption algorithm

For a tunnel, enable IPsec ESP with the encryption algorithm specified. It can be either a number or an algorithm name. Note that all IPsec tunnel properties must be specified on the same command line. To disable tunnel security, specify the value of encr_alg as none. If an

ESP authentication algorithm is specified, but the

encryption algorithm is not, the default value for the ESP encryption will be null.

It is now preferable to use the ipsecconf(1M) command when configuring a tunnel's security properties. If ipsecconf was used to set a tunnel's security properties, this keyword will not affect the tunnel.

ether [address]

If no address is given and the user is root or has sufficient privileges to open the underlying datalink, then *ficient privileges to open the underlying device, then* display the current Ethernet address information.

Otherwise, if the user is root or has sufficient privileges, set the Ethernet address of the interfaces to address. The address is an Ethernet address represented as x:x:x:x:x:x where x is a hexadecimal number between 0 and FF. Similarly, for the IPoIB (IP over InfiniBand) interfaces, the address will be 20 bytes of colon-separated hex numbers between 0 and FF.

Some, though not all, Ethernet interface cards have their own addresses. To use cards that do not have their own addresses, refer to section 3.2.3(4) of the IEEE 802.3 specification for a definition of the locally administered address space. Note that all IP interfaces in an IPMP group must have unique hardware addresses; see *in.mpathd(1M)*. *administered address space. The use of multipathing groups should be restricted to those cards with their own addresses (see MULTIPATHING GROUPS).*

-failover

Set *NOFAILOVER* on the logical interface. This makes the associated address available for use by *in.mpathd* to perform probe-based failure detection for the associated physical IP interface. As a side effect, *DEPRECATED* will also be set on the logical interface. This operation is not permitted on an IPMP IP interface. *Mark the logical interface as a non-failover interface. Addresses assigned to non-failover logical interfaces will not failover when the interface fails. Status display shows NOFAILOVER as part of flags.*

failover

Clear *NOFAILOVER* on the logical interface. This is the default. These logical interfaces are subject to migration when brought up (see IP MULTIPATHING GROUPS).
Mark the logical interface as a failover interface. An address assigned to such an interface will failover when the interface fails. Status display does not show NOFAILOVER as part of flags.

group [name | ""]
group [name |""]

When applied to a physical interface, it places the interface into the named group. If the group does not exist, it will be created, along with one or more IPMP IP interfaces (for IPv4, IPv6, or both). Any *UP* addresses that are not also marked *NOFAILOVER* are subject to migration to the IPMP IP interface (see IP MULTIPATHING GROUPS). Specifying a group name of *""* removes the physical IP interface from the group.
Insert the logical interface in the multipathing group specified by name. To delete an interface from a group,

When applied to a physical IPMP IP interface, it renames the IPMP group to have the new name. If the name already exists, or a name of *""* is specified, it fails. Renaming IPMP groups is discouraged. Instead, the IPMP IP interface should be given a meaningful name when it is created via the *ipmp* subcommand, which the system will also use as the IPMP group name.

use a null string "". When invoked on the logical interface with id zero, the status display shows the group name.

index n

Change the interface index for the interface. The value of n must be an interface index (if_index) that is not used on another interface. if_index will be a non-zero positive number that uniquely identifies the network interface on the system.

ipmp

Create an IPMP IP interface with the specified name. An interface must be separately created for use by IPv4 and IPv6. The *address_family* parameter controls whether the command applies to IPv4 or IPv6 (IPv4 if unspecified). All IPMP IP interfaces have the *IPMP* flag set.

metric n

Set the routing metric of the interface to n; if no value is specified, the default is 0. The routing metric is used by the routing protocol. Higher metrics have the effect of making a route less favorable. Metrics are counted as addition hops to the destination network or host.

modinsert mod_name@pos

Insert a module with name mod_name to the stream of the device at position pos. The position is relative to the stream head. Position 0 means directly under stream head.

Based upon the example in the modlist option, use the following command to insert a module with name ipqos under the ip module and above the firewall module:

```
example% ifconfig eri0 modinsert ipqos@2
```

A subsequent listing of all the modules in the stream of the device follows:

```
example% ifconfig eri0 modlist
0 arp
1 ip
2 ipqos
3 firewall
4 eri
```


modlist

List all the modules in the stream of the device.

The following example lists all the modules in the stream of the device:

```
example% ifconfig eri0 modlist
0 arp
1 ip
2 firewall
4 eri
```

modremove mod_name@pos

Remove a module with name mod_name from the stream of the device at position pos. The position is relative to the stream head.

Based upon the example in the modinsert option, use the following command to remove the firewall module from the stream after inserting the ipqos module:

```
example% ifconfig eri0 modremove firewall@3
```

A subsequent listing of all the modules in the stream of the device follows:

```
example% ifconfig eri0 modlist
0 arp
1 ip
2 ipqos
3 eri
```

Note that the core IP stack modules, for example, ip and tun modules, cannot be removed.

mtu n

Set the maximum transmission unit of the interface to n. For many types of networks, the mtu has an upper limit, for example, 1500 for Ethernet. This option sets the FIXEDMTU flag on the affected interface.

netmask mask

For IPv4 only. Specify how much of the address to reserve for subdividing networks into subnetworks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion. The mask can be specified in one of four ways:

1. with a single hexadecimal number with a leading 0x,
2. with a dot-notation address,
3. with a "+" (plus sign) address, or
4. with a pseudo host name/pseudo network name found in the network database networks(4).

If a "+" (plus sign) is given for the netmask value, the mask is looked up in the netmasks(4) database. This lookup finds the longest matching netmask in the database by starting with the interface's IPv4 address as the key and iteratively masking off more and more low order bits of the address. This iterative lookup ensures that the netmasks(4) database can be used to specify the netmasks when variable length subnetmasks are used within a network number.

If a pseudo host name/pseudo network name is supplied as the netmask value, netmask data may be located in the hosts or networks database. Names are looked up by first using gethostbyname(3NSL). If not found there, the names are looked up in getnetbyname(3SOCKET). These interfaces may in turn use nsswitch.conf(4) to determine what data store(s) to use to fetch the actual value.

For both inet and inet6, the same information conveyed by mask can be specified as a prefix_length attached to the address parameter.

nud

Enables the neighbor unreachability detection mechanism on a point-to-point physical interface.

-nud

Disables the neighbor unreachability detection mechanism on a point-to-point physical interface.

plumb

For a physical IP interface, open the datalink associated with the physical interface name and set up the plumbing needed for IP to use the datalink. When used with a logical interface name, this command is used to create a specific named logical interface on an

existing physical IP interface.
Open the device associated with the physical interface name and set up the streams needed for IP to use the device. When used with a logical interface name, this command is used to create a specific named logical interface. An interface must be separately plumbed for use by IPv4 and IPv6. The address_family parameter controls whether the ifconfig command applies to IPv4 or IPv6.

An interface must be separately plumbed for IPv4 and IPv6 according to the *address_family* parameter (IPv4 if unspecified). Before an interface has been plumbed, it will not be shown by *ifconfig -a*.
Before an interface has been plumbed, the interface will not show up in the output of the ifconfig -a command.

Note that IPMP IP interfaces are not tied to a specific datalink and are instead created with the *ipmp* subcommand.

private

Tells the in.routed routing daemon that a specified logical interface should not be advertised.

-private

Specify unadvertised interfaces.

removeif address

Remove the logical interface on the physical interface **specified that matches the address specified.**
specified that matches the address specified. When the interface is part of a multipathing group, the logical interface will be removed from the physical interface in the group that holds the address.

router

Enable IP forwarding on the interface. When enabled, the **interface is marked *ROUTER*, and IP packets can be forwarded to and from the interface. Enabling *ROUTER* on any IP interface in an IPMP group applies the flag to all IP interfaces in that IPMP group.**
interface is marked ROUTER, and IP packets can be forwarded to and from the interface.

-router

Disable IP forwarding on the interface. IP packets are **not forwarded to and from the interface. Disabling *ROUTER* on any IP interface in an IPMP group disables it on all IP interfaces in that IPMP group.**
not forwarded to and from the interface.

set

Set the address, prefix_length or both, for a logical interface.

standby

Mark the physical IP interface as a *STANDBY* interface. If an interface is marked *STANDBY* and is part of an IPMP group, the interface will not be used for data traffic unless another interface in the IPMP group becomes unusable. When a *STANDBY* interface is functional but not being used for data traffic, it will also be marked *INACTIVE*. This operation is not permitted on an IPMP IP interface.
Marks the physical interface as a standby interface. If the interface is marked STANDBY and is part of the multipathing group, the interface will not be selected to send out packets unless some other interface in the group has failed and the network access has been failed over to this standby interface.

The status display shows "STANDBY, INACTIVE" indicating that that the interface is a standby and is also inactive. IFF_INACTIVE will be cleared when some other interface belonging to the same multipathing group fails over to this interface. Once a failback happens, the status display will return to INACTIVE.

-standby

Clear *STANDBY* on this interface. This is the default.
Turns off standby on this interface.

subnet

Set the subnet address for an interface.

tdst tunnel_dest_address

Set the destination address of a tunnel. The address should not be the same as the dest_address of the tunnel, because no packets leave the system over such a tunnel.

thoplimit n

Set the hop limit for a tunnel interface. The hop limit value is used as the TTL in the IPv4 header for the IPv6-in-IPv4 and IPv4-in-IPv4 tunnels. For IPv6-in-IPv6 and IPv4-in-IPv6 tunnels, the hop limit value is used as the hop limit in the IPv6 header.

token address/prefix_length

Set the IPv6 token of an interface to be used for address autoconfiguration.

example% ifconfig eri0 inet6 token ::1/64

trailers

This flag previously caused a nonstandard encapsulation of IPv4 packets on certain link levels. Drivers supplied with this release no longer use this flag. It is provided for compatibility, but is ignored.

-trailers

Disable the use of a "trailer" link level encapsulation.

tsrc tunnel_src_address

Set the source address of a tunnel. This is the source address on an outer encapsulating IP header. It must be an address of another interface already configured using ifconfig.

unplumb

For a physical or IPMP interface, remove all associated logical IP interfaces and tear down any plumbing needed for IP to use the interface. For an IPMP IP interface, this command will fail if the group is not empty. For a logical interface, the logical interface is removed.
Close the device associated with this physical interface name and any streams that ifconfig set up for IP to use the device. When used with a logical interface name, the logical interface is removed from the system. After this command is executed, the device name will no longer appear in the output of ifconfig -a.

An interface must be separately unplumbed for IPv4 and IPv6 according to the *address_family* parameter (IPv4 if unspecified). Upon success, the interface name will no longer appear in the output of *ifconfig -a*.

up

Mark a logical interface *UP*. As a result, the IP module will accept packets destined to the associated address (unless the address is zero), along with any associated multicast and broadcast IP addresses. Similarly, the IP module will allow packets to be sent with the associated address as a source address.
Mark a logical interface "up". This happens automatically when assigning the first address to a logical interface. The up option enables an interface after an ifconfig down, which reinitializes the hardware.

usesrc [name | none]

Specify a physical interface to be used for source address selection. If the keyword none is used, then any previous selection is cleared.

When an application does not choose a non-zero source address using bind(3SOCKET), the system will select an appropriate source address based on the outbound interface and the address selection rules (see ipaddrsel(1M)).

When usesrc is specified and the specified interface is selected in the forwarding table for output, the system looks first to the specified physical interface and its associated logical interfaces when selecting a source address. If no usable address is listed in the forwarding table, the ordinary selection rules apply. For example, if you enter:

ifconfig eri0 usesrc vni0

...and vni0 has address 10.0.0.1 assigned to it, the system will prefer 10.0.0.1 as the source address for any packets originated by local connections that are sent through eri0. Further examples are provided in the EXAMPLES section.

While you can specify any physical interface (or even loopback), be aware that you can also specify the virtual IP interface (see vni(7D)). The virtual IP interface is not associated with any physical hardware and is thus immune to hardware failures. You can specify any number of physical interfaces to use the source address hosted on a single virtual interface. This simplifies the configuration of routing-based multipathing. If one of the physical interfaces were to fail, communication would continue through one of the remaining, functioning

physical interfaces. This scenario assumes that the reachability of the address hosted on the virtual interface is advertised in some manner, for example, through a routing protocol.

Because the ifconfig preferred option is applied to all interfaces, it is coarser-grained than the usesrc option. It will be overridden by usesrc and setsrc (route subcommand), in that order.

The use of the usesrc option is mutually exclusive of **the IPMP *group* and *standby* subcommands. That is, if an interface is already part of a IPMP group or specified as a *STANDBY* interface, then it cannot be specified with a usesrc option, and vice-versa.**
the IP multipathing ifconfig options, group and standby. That is, if an interface is already part of a IP multipathing group or specified as a standby interface, then it cannot be specified with a usesrc option, and vice-versa. For more details on IP multipathing, see in.mpathd(1M) and the .

xmit

Enable a logical interface to transmit packets. This is the default behavior when the logical interface is up.

-xmit

Disable transmission of packets on an interface. The interface will continue to receive packets.

zone zonename

Place the logical interface in zone zonename. The named zone must be active in the kernel in the ready or running state. The interface is unplumbed when the zone is halted or rebooted. The zone must be configure to be an shared-IP zone. zonecfg(1M) is used to assign network interface names to exclusive-IP zones.

-zone

Place IP interface in the global zone. This is the default.

OPERANDS

The interface operand, as well as address parameters that affect it, are described below.

interface

A string of one of the following forms:

- name physical-unit, for example, eri0 or ce1
- name physical-unit:logical-unit, for example, eri0:1
- ip.tunN or ip6.tunN, for tunnels

If the interface name starts with a dash (-), it is interpreted as a set of options which specify a set of interfaces. In such a case, -a must be part of the options and any of the additional options below can be added in any order. If one of these interface names is given, the commands following it are applied to all of the interfaces that match.

-a

Apply the command to all interfaces of the specified address family. If no address family is supplied, either on the command line or by means of /etc/default/inet_type, then all address families will be selected.

-d

Apply the commands to all "down" interfaces in the system.

-D

Apply the commands to all interfaces not under DHCP (Dynamic Host Configuration Protocol) control.

-u

Apply the commands to all "up" interfaces in the system.

-Z

Apply the commands to all interfaces in the user’s zone.

-4

Apply the commands to all IPv4 interfaces.

-6

Apply the commands to all IPv6 interfaces.

address_family

The address family is specified by the address_family parameter. The ifconfig command currently supports the following families: inet and inet6. If no address family is specified, the default is inet.

ifconfig honors the DEFAULT_IP setting in the /etc/default/inet_type file when it displays interface information . If DEFAULT_IP is set to IP_VERSION4, then

ifconfig will omit information that relates to IPv6 interfaces. However, when you explicitly specify an address family (inet or inet6) on the ifconfig command line, the command line overrides the DEFAULT_IP settings.

address

For the IPv4 family (inet), the address is either a host name present in the host name data base (see hosts(4)) or in the Network Information Service (NIS) map hosts, or an IPv4 address expressed in the Internet standard "dot notation".

For the IPv6 family (inet6), the address is either a host name present in the host name data base (see hosts(4)) or in the Network Information Service (NIS) map ipnode, or an IPv6 address expressed in the Internet standard colon-separated hexadecimal format represented as x:x:x:x:x:x:x:x where x is a hexadecimal number between 0 and FFFF.

prefix_length

For the IPv4 and IPv6 families (inet and inet6), the prefix_length is a number between 0 and the number of bits in the address. For inet, the number of bits in the

address is 32; for inet6, the number of bits in the address is 128. The prefix_length denotes the number of leading set bits in the netmask.

dest_address

If the dest_address parameter is supplied in addition to the address parameter, it specifies the address of the correspondent on the other end of a point-to-point link.

tunnel_dest_address

An address that is or will be reachable through an interface other than the tunnel being configured. This tells the tunnel where to send the tunneled packets. This address must not be the same as the interface destination address being configured.

tunnel_src_address

An address that is attached to an already configured interface that has been configured "up" with ifconfig.

INTERFACE FLAGS

The ifconfig command supports the following interface flags. The term "address" in this context refers to a logical interface, for example, eri0:0, while "interface " refers to the physical interface, for example, eri0.

ADDRCONF

The address is from stateless addrconf. The stateless mechanism allows a host to generate its own address using a combination of information advertised by routers and locally available information. Routers advertise prefixes that identify the subnet associated with the link, while the host generates an "interface identifier" that uniquely identifies an interface in a subnet. In the absence of information from routers, a host can generate link-local addresses. This flag is specific to IPv6.

ANYCAST

Indicates an anycast address. An anycast address identifies the nearest member of a group of systems that provides a particular type of service. An anycast address

is assigned to a group of systems. Packets are delivered to the nearest group member identified by the anycast address instead of being delivered to all members of the group.

BROADCAST

This broadcast address is valid. This flag and POINTTOPOINT are mutually exclusive

CoS

This interface supports some form of Class of Service (CoS) marking. An example is the 802.1D user priority **marking supported on VLAN interfaces. For IPMP IP interfaces, this will only be set if all interfaces in the group have CoS set.**
marking supported on VLAN interfaces.

DEPRECATED

This address is deprecated. This address will not be used as a source address for outbound packets unless

there are no other addresses on this interface or an application has explicitly bound to this address. An **IPv6 deprecated address is part of the standard mechanism for renumbering in IPv6 and will eventually be deleted when not used. For both IPv4 and IPv6, *DEPRECATED* is also set on all *NOFAILOVER* addresses, though this may change in a future release.**
IPv6 deprecated address will eventually be deleted when not used, whereas an IPv4 deprecated address is often used with IP network multipathing IPv4 test addresses, which are determined by the setting of the NOFAILOVER flag. Further, the DEPRECATED flag is part of the standard mechanism for renumbering in IPv6.

DHCPRUNNING

The logical interface is managed by *dhcpagent(1M)*.
DHCP

DHCP is used to manage this address.

DUPLICATE

The logical interface has been disabled because the IP address configured on the interface is a duplicate. Some

other node on the network is using this address. If the address was configured by DHCP or is temporary, the system will choose another automatically, if possible. Otherwise, the system will attempt to recover this address periodically and the interface will recover when the conflict has been removed from the network. Changing the address or netmask, or setting the logical interface to up will restart duplicate detection. Setting the interface to down terminates recovery and removes the DUPLICATE flag.

FAILED

The *in.mpathd* daemon has determined that the interface has failed. *FAILED* interfaces will not be used to send or receive IP data traffic. If this is set on a physical IP interface in an IPMP group, IP data traffic will continue to flow over other usable IP interfaces in the IPMP group. If this is set on an IPMP IP interface, the entire group has failed and no data traffic can be sent or received over any interfaces in that group.
The interface has failed. New addresses cannot be created on this interface. If this interface is part of an IP network multipathing group, a failover will occur to another interface in the group, if possible

FIXEDMTU

The MTU has been set using the -mtu option. This flag is read-only. Interfaces that have this flag set have a fixed MTU value that is unaffected by dynamic MTU changes that can occur when drivers notify IP of link MTU changes.

INACTIVE

The physical interface is functioning but is not used to send or receive data traffic according to administrative policy. This flag is initially set by the *standby* subcommand and is subsequently controlled by *in.mpathd*. It also set when *FAILBACK=no* mode is enabled (see *in.mpathd(1M)*) to indicate that the IP interface has repaired but is not being used.
Indicates that the interface is not currently being used for regular traffic by the system. New addresses cannot

IPMP

Indicates that this is an IPMP IP interface.

be created on this interface. The flag is set automatically on standby interfaces. It can also be set when the

system detects that a failed interface has been repaired and FAILBACK=no is configured in /etc/default/mpathd. The flag is cleared when the interface fails or when a failover to that interface occurs.

LOOPBACK

Indicates that this is the loopback interface.

MIP

Indicates that mobile IP controls this interface.

MULTI_BCAST

Indicates that the broadcast address is used for multicast on this interface.

MULTICAST

The interface supports multicast. IP assumes that any interface that supports hardware broadcast, or that is a point-to-point link, will support multicast.

NOARP

There is no address resolution protocol (ARP) for this interface that corresponds to all interfaces for a device without a broadcast address. This flag is specific to IPv4.

NOFAILOVER

The address associated with this logical interface is available to *in.mpathd* for probe-based failure detection of the associated physical IP interface.
This address will not failover if the interface fails. IP network multipathing test addresses must be marked nofailover.

NOLOCAL

The interface has no address , just an on-link subnet.

NONUD

NUD is disabled on this interface. NUD (neighbor unreachability detection) is used by a node to track the reachability state of its neighbors, to which the node actively sends packets, and to perform any recovery if a neighbor is detected to be unreachable. This flag is specific to IPv6.

NORTEXCH

The interface does not exchange routing information. For RIP-2, routing packets are not sent over this interface. Additionally, messages that appear to come over this interface receive no response. The subnet or address of this interface is not included in advertisements over other interfaces to other routers.

NOXMIT

Indicates that the address does not transmit packets. RIP-2 also does not advertise this address.

OFFLINE

The interface is offline and thus cannot send or receive IP data traffic. This is only set on IP interfaces in an IPMP group. See *if_mpadm(1M)* and *cfgadm(1M)*.
Indicates that the interface has been offlined. New addresses cannot be created on this interface. Interfaces in an IP network multipathing group are offlined prior to removal and replacement using dynamic reconfiguration.

POINTOPOINT

Indicates that the address is a point-to-point link. This flag and BROADCAST are mutually exclusive

PREFERRED

This address is a preferred IPv6 source address. This address will be used as a source address for IPv6 communication with all IPv6 destinations, unless another address on the system is of more appropriate scope. The DEPRECATED flag takes precedence over the PREFERRED flag.

PRIVATE

Indicates that this address is not advertised. For RIP-2, this interface is used to send advertisements. However, neither the subnet nor this address are included in advertisements to other routers.

ROUTER

Indicates that IP packets can be forwarded to and from the interface.

RUNNING

Indicates that the required resources for an i nterface
Indicates that the required resources for an interface
are allocated. For some interfaces this also indicates
that the link is up. For IPMP IP interfaces, *RUNNING* is set as long as one IP interface in the group is active.
that the link is up.

STANDBY

Indicates that this physical interface will not be used for data traffic unless another interface in the IPMP group becomes unusable. The *INACTIVE* and *FAILED* flags indicate whether it is actively being used.
Indicates that this is a standby interface to be used on failures. Only interfaces in an IP network multipathing group should be designated as standby interfaces. If this interface is part of a IP network multipathing group, the interface will not be selected to send out packets unless some other interface in the group fails over to it.

TEMPORARY

Indicates that this is a temporary IPv6 address as defined in RFC 3041.

UNNUMBERED

This flag is set when the local IP address on the link matches the local address of some other link in the system

UP

Indicates that the logical interface (and the associated physical interface) is up. The IP module will accept packets destined to UP addresses (unless the address is zero), along with any associated multicast and broadcast IP addresses. Similarly, the IP module will allow packets to be sent with an UP address as a source address.

Indicates that the interface is up, that is, all the routing entries and the like for this interface have been set up.

VIRTUAL

Indicates that the physical interface has no underlying hardware. It is not possible to transmit or receive packets through a virtual interface. These interfaces are useful for configuring local addresses that can be
used on multiple interfaces. (See also the *usesrc*
used on multiple interfaces. (See also the -usesrc
option.)

XRESOLV

Indicates that the interface uses an IPv6 external resolver.

LOGICAL INTERFACES

Solaris TCP/IP allows multiple logical interfaces to be associated with a physical network interface. This allows a single machine to be assigned multiple IP addresses, even though it may have only one network interface. Physical network interfaces have names of the form driver-name physical-unit-number, while logical interfaces have names of the form driver-name physical-unit-number:logical-unit-number. A physical interface is configured into the system using the plumb command. For example:

```
example% ifconfig eri0 plumb
```

Once a physical interface has been "plumbed", logical interfaces associated with the physical interface can be configured by separate -plumb or -addif options to the ifconfig command.

```
example% ifconfig eri0:1 plumb
```

allocates a specific logical interface associated with the physical interface eri0. The command

```
example% ifconfig eri0 addif 192.168.200.1/24 up
```

allocates the next available logical unit number on the eri0 physical interface and assigns an address and prefix_length.

A logical interface can be configured with parameters (address,prefix_length, and so on) different from the physical interface with which it is associated. Logical interfaces that are associated with the same physical interface can be given different parameters as well. Each logical interface must be associated with an existing and "up" physical interface. So, for example, the logical interface eri0:1 can only be configured after the physical interface eri0 has been plumbed.

To delete a logical interface, use the *unplumb* or *removeif* options. For example,

To delete a logical interface, use the -unplumb or -removeif options. For example,

```
example% ifconfig eri0:1 down unplumb
```

will delete the logical interface *eri0:1*.

IP MULTIPATHING GROUPS

Physical interfaces that share the same IP broadcast domain _must_ be collected into a single IP Multipathing (IPMP) group using the *group* subcommand. Each IPMP group has an associated IPMP IP interface, which can either be explicitly created (the preferred method) by using the *ipmp* subcommand or implicitly created by *ifconfig* in response to placing an IP interface into a new IPMP group. Implicitly-created IPMP interfaces will be named ipmp_N_ where _N_ is the lowest integer that doesn't conflict with an existing IP interface name or IPMP group name.

Each IPMP IP interface is created with a matching IPMP group name, though it can be changed using the *group* subcommand. Each IPMP IP interface hosts a set of highly-available IP addresses. These addresses will remain reachable so long as at least one interface in the group is active, where "active" is defined as having at least one UP address and having *INACTIVE*, *FAILED*, and *OFFLINE* clear. IP addresses hosted on the IPMP IP interface may either be configured statically or configured through DHCP via the *dhcp* subcommand.

will delete the logical interface eri0:1.

Interfaces assigned to the same IPMP group are treated as

equivalent and monitored for failure by *in.mpathd*. Provided that active interfaces in the group remain, IP interface failures (and any subsequent repairs) are handled transparently to sockets-based applications. IPMP is also integrated with the Dynamic Reconfiguration framework (see *cfgadm(1M)*), which enables network adapters to be replaced transparently to sockets-based applications.

MULTIPATHING GROUPS

Physical interfaces that share the same IP broadcast domain can be collected into a multipathing group using the group keyword. Interfaces assigned to the same multipathing group are treated as equivalent and outgoing traffic is spread across the interfaces on a per-IP-destination basis. In addition, individual interfaces in a multipathing group are monitored for failures; the addresses associated with failed interfaces are automatically transferred to other functioning interfaces within the group.

The IP module automatically load-spreads all outbound traffic across all active interfaces in an IPMP group. Similarly, all *UP* addresses hosted on the IPMP IP interface and will be distributed across the active interfaces to promote inbound load-spreading. The *ipmpstat(1M)* utility allows many aspects of the IPMP subsystem to be observed, including the current binding of IP data addresses to IP interfaces.

When an interface is placed into an IPMP group, any *UP* logical interfaces are "migrated" to the IPMP IP interface for use by the group, unless:

For more details on IP multipathing, see in.mpathd(1M) and the . See netstat(1M) for per-IP-destination information.

*** The logical interface is marked *NOFAILOVER***
*** The logical interface hosts an IPv6 link-local address.**
*** The logical interface hosts an IPv4 0.0.0.0 address.**

Likewise, once an interface is in a group, if changes are made to a logical interface such that it is *UP* and not exempted by one of the conditions above, it will also migrate to the associated IPMP IP interface. Logical interfaces never migrate back, even if the physical interface that contributed the address is removed from the group.

Each interface placed into an IPMP group may be optionally configured with a "test" address that *in.mpathd* will use for probe-based failure detection; see *in.mpathd(1M)*. These addresses must be marked *NOFAILOVER* (using the *-failover* subcommand) prior to being marked *UP*. Test addresses may also be acquired through DHCP via the *dhcp* subcommand.

For more background on IPMP, please see the "IPMP Administrative Overview" and "IPMP Configuration Tasks" chapters of the administrator documentation.

CONFIGURING IPV6 INTERFACES

When an IPv6 physical interface is plumbed and configured "up" with ifconfig, it is automatically assigned an IPv6 link-local address for which the last 64 bits are calculated from the MAC address of the interface.

```
example% ifconfig eri0 inet6 plumb up
```

The following example shows that the link-local address has a prefix of fe80::/10.

```
example% ifconfig eri0 inet6
ce0: flags=2000841<UP,RUNNING,MULTICAST,IPv6>
```



```
mtu 1500 index 2
inet6 fe80::a00:20ff:fe8e:f3ad/10
```

Link-local addresses are only used for communication on the local subnet and are not visible to other subnets.

If an advertising IPv6 router exists on the link advertising prefixes, then the newly plumbed IPv6 interface will auto-configure logical interface(s) depending on the prefix advertisements. For example, for the prefix advertisement 2001:0db8:3c4d:0:55::/64, the autoconfigured interface will look like:

```
eri0:2: flags=2080841<UP,RUNNING,MULTICAST,ADDRCONF,IPv6>
mtu 1500 index 2
inet6 2001:0db8:3c4d:55:a00:20ff:fe8e:f3ad/64
```

Even if there are no prefix advertisements on the link, you can still assign global addresses manually, for example:

```
example% ifconfig eri0 inet6 addif \
2001:0db8:3c4d:55:a00:20ff:fe8e:f3ad/64 up
```

To configure boot-time defaults for the interface eri0, place the following entry in the /etc/hostname6.eri0 file:

```
addif 2001:0db8:3c4d:55:a00:20ff:fe8e:f3ad/64 up
```

Configuring IPv6/IPv4 tunnels

An IPv6 over IPv4 tunnel interface can send and receive IPv6 packets encapsulated in an IPv4 packet. Create tunnels at both ends pointing to each other. IPv6 over IPv4 tunnels require the tunnel source and tunnel destination IPv4 and

IPv6 addresses. Solaris 8 supports both automatic and configured tunnels. For automatic tunnels, an IPv4-compatible IPv6 address is used. The following demonstrates auto-tunnel configuration:

```
example% ifconfig ip.atun0 inet6 plumb
example% ifconfig ip.atun0 inet6 tsrc IPv4-address \
::IPv4 address/96 up
```


where IPv4-address is the IPv4 address of the interface through which the tunnel traffic will flow, and IPv4-address, ::<IPv4-address>, is the corresponding IPv4-compatible IPv6 address.

The following is an example of a configured tunnel:

```
example% ifconfig ip.tun0 inet6 plumb tsrc my-ipv4-address \
tdst peer-ipv4-address up
```

This creates a configured tunnel between my-ipv4-address and peer-ipv4-address with corresponding link-local addresses. For tunnels with global or site-local addresses, the logical tunnel interfaces need to be configured in the following form:

```
example% ifconfig ip.tun0 inet6 addif my-v6-address peer-v6-address up
```

For example,

```
example% ifconfig ip.tun0 inet6 plumb tsrc 109.146.85.57 \
tdst 109.146.85.212 up
example% ifconfig ip.tun0 inet6 addif 2::45 2::46 up
```

To show all IPv6 interfaces that are up and configured:

```
example% ifconfig -au6
ip.tun0: flags=2200851<UP,POINTOPOINT,RUNNING,MULTICAST,NONUD,IPv6>
mtu 1480 index 3
inet tunnel src 109.146.85.57 tunnel dst 109.146.85.212
tunnel security settings --> use 'ipsecconf -ln -i ip.tun1'
tunnel hop limit 60
inet6 fe80::6d92:5539/10 --> fe80::6d92:55d4
ip.tun0:1: flags=2200851<UP,POINTOPOINT,RUNNING,MULTICAST,NONUD,IPv6>
```

```
1795                mtu 1480 index 3
1796              inet6 2::45/128 --> 2::46
```

```
1801      In the output above, note the line that begins with  "tunnel
1802      security   settings".   The  content  of  this  line  varies
```



```
1817      according to whether and how you have set your security set-
1818      tings. See "Display of Tunnel Security Settings," below.

1820   Configuring IPv4/IPv6 Tunnels
1821      An IPv4 over IPv6 tunnel interface can send and receive IPv4
1822      packets  encapsulated  in  an IPv6 packet. Create tunnels at
1823      both ends pointing to each other.  IPv4  over  IPv6  tunnels
1824      require  the  tunnel  source and tunnel destination IPv6 and
1825      IPv4 addresses. The following demonstrates auto-tunnel  con-
1826      figuration:

1828        example% ifconfig ip6.tun0 inet plumb tsrc my-ipv6-address \
1829          tdst peer-ipv6-address my-ipv4-address \
1830          peer-ipv4-address up
```

```
1835      This creates a configured tunnel between my-ipv6-address and
1836      peer-ipv6-address with my-ipv4-address and peer-ipv4-address
1837      as the endpoints of the point-to-point interface, for  exam-
1838      ple:

1840        example% ifconfig ip6.tun0 inet plumb tsrc fe80::1 tdst fe80::2 \
1841        10.0.0.208 10.0.0.210 up
```

```
1846      To show all IPv4 interfaces that are up and configured:

1848        example% ifconfig -au4
1849        lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
1850            inet 127.0.0.1 netmask ff000000
1851        eri0: flags=1004843<UP,BROADCAST,RUNNING,MULTICAST,DHCP,IPv4> mtu 1500 \
1852        index 2
1853            inet 172.17.128.208 netmask ffffff00 broadcast 172.17.128.255
1854        ip6.tun0: flags=10008d1<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST,IPv4> \
1855        mtu 1460
1856            index 3
1857            inet6 tunnel src fe80::1 tunnel dst fe80::2
1858            tunnel security settings  -->  use 'ipsecconf -ln -i ip.tun1'
1859            tunnel hop limit 60 tunnel encapsulation limit 4
1860            inet 10.0.0.208 --> 10.0.0.210 netmask ff000000
```



```
1865      In the output above, note the line that begins with  "tunnel
1866      security  settings". The content of this line varies accord-
1867      ing to whether and how you have set your security  settings.
1868      See "Display of Tunnel Security Settings," below.
```



```
1883   Display of Tunnel Security Settings
1884      The ifconfig output for tunneled interfaces indicates  secu-
1885      rity  settings, if present, for a tunnel. The content of the
1886      line showing your settings differs depending on how you have
1887      made your settings:

1889          o    If you set your security policy using the  ifconfig
1890               -auth_algs, -encr_algs, and -encr_auth_algs options
1891               and do not  use  ipsecconf(1M),  ifconfig  displays
1892               your settings for each of these options.

1894          o    If you set your security policy using ipsecconf(1M)
1895               with  the  tunnel  keyword  (the preferred method),
1896               ifconfig displays:

1898                 tunnel security settings  -->  use 'ipsecconf -ln -i ip.tun1'


1901               ...in  effect,  hiding  your  settings  from  those
1902               without privileges to view them.

1904               If you do net set  security  policy,  using  either
1905               ifconfig  or ipsecconf, there is no tunnel security
1906               setting displayed.

1908 EXAMPLES
1909      Example 1 Using the ifconfig Command


1912      If your workstation is not attached to an Ethernet, the net-
1913      work  interface,  for example, eri0, should be marked "down"
1914      as follows:


1917        example% ifconfig eri0 down


1921      Example 2 Printing Addressing Information


1924      To print out the addressing information for each  interface,
1925      use the following command:
```

```
1928        example% ifconfig -a


1932      Example 3 Resetting the Broadcast Address



1938 SunOS 5.11          Last change: 21 Jan 2007                  29



1945 System Administration Commands                        ifconfig(1M)


1949      To reset each interface's broadcast address after  the  net-
1950      masks have been correctly set, use the next command:

1953        example% ifconfig -a broadcast +


1957      Example 4 Changing the Ethernet Address

1960      To change the Ethernet address for interface  ce0,  use  the
1961      following command:

1964        example% ifconfig ce0 ether aa:1:2:3:4:5


1968      Example 5 Configuring an IP-in-IP Tunnel

1971      To configure an IP-in-IP tunnel, first  plumb  it  with  the
1972      following command:

1975        example% ifconfig ip.tun0 plumb



1980      Then configure it as a point-to-point  interface,  supplying
1981      the tunnel source and the tunnel destination:

1984        example% ifconfig ip.tun0 myaddr mydestaddr tsrc another_myaddr \
1985                  tdst a_dest_addr up



1990      Use ipsecconf(1M), as described above, to  configure  tunnel
1991      security properties.
```



```
1994      Example 6 Configuring 6to4 Tunnels

1997      To configure 6to4 tunnels, use the following commands:

2000        example% ifconfig ip.6to4tun0 inet6 plumb


2004 SunOS 5.11          Last change: 21 Jan 2007                  30



2011 System Administration Commands                        ifconfig(1M)


2015        example% ifconfig ip.6to4tun0 inet6 tsrc IPv4-address 6to4-address/64 up



2020      IPv4-address denotes the address of the encapsulating inter-
2021      face.  6to4-address  denotes  the  address of the local IPv6
2022      address of form 2002:IPv4-address:SUBNET-ID:HOSTID.


2026      The long form should be used to resolve any  potential  con-
2027      flicts that might arise if the system administrator utilizes
2028      an addressing plan where the values for SUBNET-ID or  HOSTID
2029      are reserved for something else.


2033      After the interface is plumbed, a 6to4 tunnel can be config-
2034      ured as follows:

2037        example% ifconfig ip.6to4tun0 inet6 tsrc IPv4-address up



2042      This short form sets the address. It uses the convention:

2045        2002:IPv4-address::1


2049      The SUBNET-ID is 0, and the HOSTID is 1.

2052      Example 7 Configuring IP Forwarding on an Interface

2055      To enable IP forwarding on a single interface, use the  fol-
2056      lowing command:
```

```
2059        example% ifconfig eri0 router



2064      To disable IP forwarding on a single interface, use the fol-
2065      lowing command:



2070 SunOS 5.11          Last change: 21 Jan 2007                  31




2077 System Administration Commands                        ifconfig(1M)


2081        example% ifconfig eri0 -router


2085      Example 8 Configuring Source Address Selection Using a  Vir-
2086      tual Interface


2089      The following command configures  source  address  selection
2090      such  that  every  packet  that is locally generated with no
2091      bound source address and going out on qfe2 prefers a  source
2092      address hosted on vni0.


2095        example% ifconfig qfe2 usesrc vni0



2100      The ifconfig -a output for  the  qfe2  and  vni0  interfaces
2101      displays as follows:


2104        qfe2: flags=1100843<UP,BROADCAST,RUNNING,MULTICAST,ROUTER,IPv4> mtu
2105         1500 index 4
2106         usesrc vni0
2107         inet 1.2.3.4 netmask ffffff00 broadcast 1.2.3.255
2108         ether 0:3:ba:17:4b:e1
2109        vni0: flags=20011100c1<UP,RUNNING,NOARP,NOXMIT,ROUTER,IPv4,VIRTUAL>
2110         mtu 0 index 5
2111         srcof qfe2
2112         inet 3.4.5.6 netmask ffffffff



2116      Observe, above, the usesrc and srcof keywords in the  ifcon-
2117      fig  output.  These  keywords  also  appear  on  the logical
2118      instances of the physical interface, even though this  is  a
2119      per-physical  interface parameter. There is no srcof keyword
2120      in ifconfig for configuring interfaces. This information  is
2121      determined  automatically  from  the  set of interfaces that
2122      have usesrc set on them.
```



```
2126      The following command, using the none  keyword,  undoes  the
2127      effect of the preceding *ifconfig* *usesrc* command.
2100      effect of the preceding ifconfig usersrc command.


2130        example% ifconfig qfe2 usesrc none



2136 SunOS 5.11          Last change: 21 Jan 2007                  32




2143 System Administration Commands                        ifconfig(1M)


2147      Following this command, ifconfig -a output displays as  fol-
2148      lows:


2151        qfe2: flags=1100843<UP,BROADCAST,RUNNING,MULTICAST,ROUTER,IPv4> mtu
2152         1500 index 4
2153         inet 1.2.3.4 netmask ffffff00 broadcast 1.2.3.255
2154         ether 0:3:ba:17:4b:e1
2155        vni0: flags=20011100c1<UP,RUNNING,NOARP,NOXMIT,ROUTER,IPv4,VIRTUAL>
2156         mtu 0 index 5
2157         inet 3.4.5.6 netmask ffffffff


2161      Note the absence of the usesrc and  srcof  keywords  in  the
2162      output above.


2165      Example 9 Configuring Source Address Selection for  an  IPv6
2166      Address


2169      The following command configures  source  address  selection
2170      for  an  IPv6  address, selecting a source address hosted on
2171      vni0.


2174        example% ifconfig qfe1 inet6 usesrc vni0



2179      Following this command, ifconfig -a output displays as  fol-
2180      lows:


2183        qfe1: flags=2000841<UP,RUNNING,MULTICAST,IPv6> mtu 1500 index 3
2184         usesrc vni0
2185         inet6 fe80::203:baff:fe17:4be0/10
2186         ether 0:3:ba:17:4b:e0
2187        vni0: flags=2002210041<UP,RUNNING,NOXMIT,NONUD,IPv6,VIRTUAL> mtu 0
2188         index 5
2189         srcof qfe1
```

```
2190         inet6 fe80::203:baff:fe17:4444/128
2191        vni0:1: flags=2002210040<RUNNING,NOXMIT,NONUD,IPv6,VIRTUAL> mtu 0
2192         index 5
2193         srcof qfe1
2194         inet6 fec0::203:baff:fe17:4444/128
2195        vni0:2: flags=2002210040<RUNNING,NOXMIT,NONUD,IPv6,VIRTUAL> mtu 0
2196         index 5
2197         srcof qfe1
2198         inet6 2000::203:baff:fe17:4444/128


2202 SunOS 5.11          Last change: 21 Jan 2007                  33




2209 System Administration Commands                        ifconfig(1M)


2213      Depending on the scope of  the  destination  of  the  packet
2214      going  out  on qfe1, the appropriately scoped source address
2215      is selected from vni0 and its aliases.


2218      Example 10 Using Source  Address  Selection  with  Shared-IP
2219      Zones

2222      The following is an example of how the usesrc feature can be
2223      used  with  the  zones(5) facility in Solaris. The following
2224      commands are invoked in the global zone:


2227        example% ifconfig hme0 usesrc vni0
2228        example% ifconfig eri0 usesrc vni0
2229        example% ifconfig qfe0 usesrc vni0



2234      Following the preceding commands, the ifconfig -a output for
2235      the virtual interfaces would display as:


2238        vni0: flags=20011100c1<UP,RUNNING,NOARP,NOXMIT,ROUTER,IPv4,VIRTUAL>
2239          mtu 0 index 23
2240          srcof hme0 eri0 qfe0
2241          inet 10.0.0.1 netmask ffffffff
2242        vni0:1:
2243          flags=20011100c1<UP,RUNNING,NOARP,NOXMIT,ROUTER,IPv4,VIRTUAL> mtu 0
2244          index 23
2245          zone test1
2246          srcof hme0 eri0 qfe0
2247          inet 10.0.0.2 netmask ffffffff
2248        vni0:2:
2249          flags=20011100c1<UP,RUNNING,NOARP,NOXMIT,ROUTER,IPv4,VIRTUAL> mtu 0
2250          index 23
2251          zone test2
2252          srcof hme0 eri0 qfe0
2253          inet 10.0.0.3 netmask ffffffff
2254        vni0:3:
2255          flags=20011100c1<UP,RUNNING,NOARP,NOXMIT,ROUTER,IPv4,VIRTUAL> mtu 0
```



```
2256          index 23
2257          zone test3
2258          srcof hme0 eri0 qfe0
2259          inet 10.0.0.4 netmask ffffffff



2263      There is one virtual interface alias per zone (test1, test2,
2264      and  test3).  A  source  address  from the virtual interface


2268 SunOS 5.11          Last change: 21 Jan 2007                  34




2275 System Administration Commands                        ifconfig(1M)


2279      alias in the same zone is selected.  The  virtual  interface
2280      aliases were created using zonecfg(1M) as follows:


2283        example% zonecfg -z test1
2284        zonecfg:test1> add net
2285        zonecfg:test1:net> set physical=vni0
2286        zonecfg:test1:net> set address=10.0.0.2



2291      The test2  and  test3  zone  interfaces  and  addresses  are
2292      created in the same way.


2295      Example 11 Turning Off DHCPv6


2298      The following example shows how to disable automatic use  of
2299      DHCPv6  on  all interfaces, and immediately shut down DHCPv6
2300      on  the  interface   named   hme0.   See   in.ndpd(1M)   and
2301      ndpd.conf(4)  for  more  information on the automatic DHCPv6
2302      configuration mechanism.


2305        example% echo ifdefault StatefulAddrConf false >> /etc/inet/ndpd.conf
2306        example% pkill -HUP -x in.ndpd
2307        example% ifconfig hme0 dhcp release



2311 FILES
2312      /etc/netmasks

2314          Netmask data.


2317      /etc/default/inet_type

2319          Default Internet protocol type.
```

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability for command-line options	Committed
Interface Stability for command output	Uncommitted

SEE ALSO
dhcpinfo(1), dhcpagent(1M), in.mpathd(1M), in.ndpd(1M), **in.routed(1M), ipmpstat(1M), ipsecconf(1M), netstat(1M),** *in.routed(1M), ipsecconf(1M), ndd(1M), netstat(1M),* zoneadm(1M), zonecfg(1M), ethers(3SOCKET), gethostbyname(3NSL), getnetbyname(3SOCKET), hosts(4), inet_type(4), ndpd.conf(4), netmasks(4), networks(4), nsswitch.conf(4), attributes(5), privileges(5), zones(5), arp(7P), ipsecah(7P), ipsecesp(7P), tun(7M)

DIAGNOSTICS
ifconfig sends messages that indicate if:

- the specified interface does not exist
- the requested address is unknown
- the user is not privileged and tried to alter an interface’s configuration

NOTES
Do not select the names broadcast, down, private, trailers, up or other possible option names when you choose host names. If you choose any one of these names as host names, it can cause unusual problems that are extremely difficult to diagnose.

```
**********************************************************
   13177 Fri Dec 19 04:18:55 2008
new/./in.mpathd.1m.txt
Clearview IPMP manpages
**********************************************************
```

```
   1 System Administration Commands                    in.mpathd(1M)



   5 NAME
   6      in.mpathd - IP multipathing daemon
   6      in.mpathd - daemon for network adapter (NIC) failure  detec-
   7      tion, recovery, automatic failover and failback

   8 SYNOPSIS
   9      /usr/lib/inet/in.mpathd


  12 DESCRIPTION
  14      The in.mpathd daemon performs Network Interface  Card  (NIC)
  15      failure and repair detection. In the event of a NIC failure,
  16      it causes IP network access from the failed NIC to  failover
  17      to  a  standby  NIC,  if available, or to any another opera-
  18      tional NIC that has been configured as part of the same net-
  19      work  multipathing  group.  Once the failed NIC is repaired,
  20      all network access is restored to the repaired NIC.

  14      The *in.mpathd* daemon performs failure and repair detection
  15      for IP  interfaces that have been placed  into an IPMP group
  16      (or optionally, for  all IP interfaces  on the  system).  It
  17      also  controls which  IP  interfaces in  an IPMP group  are
  18      "active" (being used  by  the system  to send  or receive IP
  19      data traffic) in  a  manner which  is  consistent  with the
  20      administrator's configured policy.


  23      The *in.mpathd* daemon can  detect IP interface  failure and
  24      repair through  two methods: by monitoring the *IFF_RUNNING*
  25      flag for each  IP interface (link-based failure  detection),
  26      and  by sending and  receiving  ICMP probes  on each  IP
  27      interface  (probe-based  failure detection).  Link-based
  28      failure detection  is instantaneous and is  always enabled
  29      (provided the  network  driver  supports  the  feature);
  23      The in.mpathd daemon  can  detect  NIC  failure  and  repair
  24      through  two methods: by monitoring the IFF_RUNNING flag for
  25      each NIC (link-based failure detection), and by sending  and
  26      receiving  ICMP  echo  requests  and  replies  on  each  NIC
  27      (probe-based failure detection). Link-based  failure  detec-
  28      tion  requires  no explicit configuration and thus is always
  29      enabled (provided the  NIC  driver  supports  the  feature);
  30      probe-based  failure detection must  be enabled  through the
  31      configuration of  one or  more test  addresses  (described
  32      below),  but tests the entire  IP interface send and receive
  33      path.  The *ipmpstat(1M)* utility can be used to check which
  34      failure detection methods are enabled.
  32      below),  but  has the benefit of testing the entire NIC send
  33      and receive path.


  37      If only link-based failure detection is  enabled,  then  the
  38      health  of the interface is determined solely from the state
  39      of the IFF_RUNNING flag. Otherwise, the  interface  is  con-
  40      sidered  failed  if  either  of  the  two methods indicate a
  41      failure, and repaired once both methods indicate the failure
  42      has been corrected. Not all interfaces in a group need to be
  43      configured with the same failure detection methods.
```



```
  46      As mentioned above, to perform probe-based failure detection
  47      *in.mpathd* requires a test address on each IP interface for
  48      the  purpose of sending and  receiving probes.  Each address
  49      must be  marked  *NOFAILOVER* (see  *ifconfig(1M)*)  and
  50      *in.mpathd*  will be limited to probing  targets on the same
  51      subnet.  Each address  may  be configured  statically  or
  52      acquired via  DHCP.  To  find targets, *in.mpathd*  first
  53      consults  the routing table for  routes  on the same subnet,
  54      and uses  the  specified next-hop.  If no routes  match, it
  55      sends  all-hosts ICMP probes and selects  a  subset of the
  56      systems  that  respond.  Thus, for probe-based  failure
  57      detection to operate, there must be at least one neighbor on
  58      each subnet that responds to ICMP echo  request probes.  The
  59      *ipmpstat(1M)*  utility can be  used  to display both  the
  60      current  probe target information  and  the status of  sent
  61      probes.
  45      As mentioned above, in order to perform probe-based  failure
  46      detection in.mpathd needs a special test address on each NIC
  47      for the purpose of sending and receiving probes on the  NIC.
  48      Use the ifconfig command -failover option to configure these
  49      test addresses. See  ifconfig(1M).  The  test  address  must
  50      belong to a subnet that is known to the hosts and routers on
  51      the link.
  62

  64      Both IPv4 and  IPv6 are  supported.  If  an IP interface  is
  65      plumbed for IPv4 and an IPv4 test address is configured then
  66      *in.mpathd*  will start sending ICMPv4  probes  over that IP
  67      interface.  Similarly,  if  an IP  interface  is plumbed for
  68      IPv6 and an IPv6 test address is configured then *in.mpathd*
  69      will start  sending  ICMPv6 probes over  that IP interface.
  70      However,  note  that *in.mpathd*  will ignore  IPv6 test
  71      addresses that are not  link-local.  If  both IPv4 and  IPv6
  72      are plumbed,  it is sufficient to  configure only one of the
  73      two, that is, either  an IPv4 test address  or an IPv6  test
  74      address.  If both  IPv4  and IPv6  test addresses are
  75      configured, *in.pathd* probes using both ICMPv4 and ICMPv6.
  54      The in.mpathd daemon can detect NIC failure  and  repair  by
  55      two methods, by sending and receiving ICMP echo requests and
  56      replies on each NIC, and by monitoring the IFF_RUNNING  flag


  78      As mentioned  above,  *in.mpathd* also  controls  which IP
  79      interfaces in an IPMP group are "active" (used by the system
  80      to send  and  receive  IP  data  traffic).  Specifically,
  81      *in.mpathd* tracks the  administrative configuration of each
  82      IPMP  group and attempts to  keep  the  number of active  IP
  83      interfaces in each group consistent with that configuration.
  84      Therefore, if an active IP interface fails, *in.mpathd* will
  85      activate an *INACTIVE* interface in  the group, provided one
  86      exists  (it will prefer  *INACTIVE* interfaces that are also
  87      marked *STANDBY*).  Likewise, if an IP interface repairs and
  88      the resulting repair leaves  the IPMP group with more active
  89      interfaces than  the administrative configuration specifies,
  90      *in.mpathd*  will  deactivate one  of  the interfaces
  91      (preferably  one  marked *STANDBY*),  except  when  the
  92      *FAILBACK* variable  is used, as  described below.  Similar
  93      adjustments will be  made by  *in.mpathd* when offlining  IP
  94      interfaces (for instance, in response to *if_mpadm(1M)*).

  60 SunOS 5.11          Last change: 8 Sep 2006                  1
```

for each NIC. The link state on some models of NIC is indicated by the IFF_RUNNING flag, allowing for faster failure detection when the link goes down. The in.mpathd daemon considers a NIC to have failed if either of the above two methods indicates failure. A NIC is considered to be repaired only if both methods indicate the NIC is repaired.

The in.mpathd daemon sends the ICMP echo request probes to on-link routers. If no routers are available, it sends the probes to neighboring hosts. Thus, for network failure detection and repair, there must be at least one neighbor on each link that responds to ICMP echo request probes.

in.mpathd works on both IPv4 and IPv6. If IPv4 is plumbed on a NIC, an IPv4 test address is configured on theNIC, and the NIC is configured as part of a network multipathing group, then in.mpathd will start sending ICMP probes on the NIC using IPv4.

In the case of IPv6, the link-local address must be configured as the test address. The in.mpathd daemon will not accept a non-link-local address as a test address. If the NIC is part of a multipathing group, and the test address has been configured, then in.mpathd will probe the NIC for failures using IPv6.

Even if both the IPv4 and IPv6 protocol streams are plumbed, it is sufficient to configure only one of the two, that is, either an IPv4 test address or an IPv6 test address on a NIC. If only an IPv4 test address is configured, it probes using only ICMPv4. If only an IPv6 test address is configured, it probes using only ICMPv6. If both type test addresses are configured, it probes using both ICMPv4 and ICMPv6.

The in.mpathd daemon accesses three variable values in /etc/default/mpathd: FAILURE_DETECTION_TIME, FAILBACK and TRACK_INTERFACES_ONLY_WITH_GROUPS.

The *FAILURE_DETECTION_TIME* variable specifies the probe-based failure detection time. The shorter the failure detection time, the more probe traffic. The default value of *FAILURE_DETECTION_TIME* is 10 seconds. This means that IP interface failure will be detected by *in.mpathd* within 10 seconds. The IP interface repair detection time is always twice the value of *FAILURE_DETECTION_TIME*. Note that failures and repairs detected by link-based failure detection are acted on immediately, though *in.mpathd* may ignore link state changes if it suspects that the link state is flapping due to defective hardware; see DIAGNOSTICS.

The FAILURE_DETECTION_TIME variable specifies the NIC failure detection time for the ICMP echo request probe method of detecting NIC failure. The shorter the failure

detection time, the greater the volume of probe traffic. The default value of FAILURE_DETECTION_TIME is 10 seconds. This means that NIC failure will be detected by in.mpathd within 10 seconds. NIC failures detected by the IFF_RUNNING flag

By default, *in.mpathd* limits failure and repair detection to IP interfaces that are configured as part of a named IPMP group. Setting *TRACK_INTERFACES_ONLY_WITH_GROUPS* to *no* enables failure and repair detection on all IP interfaces, even if they are not part of a named IPMP group. IP interfaces that are tracked but not part of a named IPMP group are considered to be part of the "anonymous" IPMP group. In addition to having no name, this IPMP group is special in that its IP interfaces are not equivalent and thus cannot take over for one another in the event of an IP interface failure. That is, the anonymous IPMP group can only be used for failure and repair detection, and provides no high-availability or load-spreading.

As described above, when *in.mpathd* detects that an IP interface has repaired, it activates it so that it will again be used to send and receive IP data traffic. However, if *FAILBACK* is set to *no*, then the IP interface will only be activated if no other active IP interfaces in the group remain. However, the interface may subsequently be activated if another IP interface in the group fails.

being cleared are acted on as soon as the in.mpathd daemon notices the change in the flag. The NIC repair detection time cannot be configured; however, it is defined as double the value of FAILURE_DETECTION_TIME.

By default, in.mpathd does failure detection only on NICs that are configured as part of a multipathing group. You can set TRACK_INTERFACES_ONLY_WITH_GROUPS to no to enable failure detection by in.mpathd on all NICs, even if they are not part of a multipathing group. However, in.mpathd cannot do failover from a failed NIC if it is not part of a multipathing group.

The in.mpathd daemon will restore network traffic back to the previously failed NIC, after it has detected a NIC repair. To disable this, set the value of FAILBACK to no in /etc/default/mpathd.

FILES

/etc/default/mpathd Contains default values used by the in.mpathd daemon.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr

SEE ALSO

ifconfig(1M), ipmpstat(1M), if_mpadm(1M), icmp(7P), icmp6(7P)
ifconfig(1M), attributes(5), icmp(7P), icmp6(7P),

DIAGNOSTICS

IP interface *interface_name* has a hardware address which is not unique in group *group_name*; offlining
Test address address is not unique; disabling probe based failure detection on interface_name
Description:

For probe-based failure detection, load-spreading, and other code IPMP features to work properly, each IP interface in an IPMP group must have a unique hardware address. If this requirement is not met, *in.mpathd* will automatically offline all but one of the IP interfaces with duplicate hardware addresses.

IP interface *interface_name* now has a unique hardware address in group *group_name*; onlining
Description:

The previously-detected duplicate hardware address is now unique, and therefore *in.mpathd* has brought *interface_name* back online.

Test address *address* is not unique in group; disabling probe-based failure detection on *interface_name*
Description:

For in.mpathd to perform probe-based failure detection, **each test address in the group must be unique.** *each test address in the group must be unique. Since the IPv6 test address is a link-local address derived from the MAC address, each IP interface in the group must*

No test address configured on interface *interface_name*; disabling probe-based failure detection on it
Description:
have a unique MAC address.

For *in.mpathd* to perform probe-based failure detection on an IP interface, it must be configured with a test address: IPv4, IPv6, or both.

NIC interface_name of group group_name is not plumbed for IPv[4|6] and may affect failover capability
Description:

All NICs in a multipathing group must be homogeneously plumbed. For example, if a NIC is plumbed for IPv4, then all NICs in the group must be plumbed for IPv4. The **STREAMS modules pushed on all NICs must also be identical.**
streams modules pushed on all NICs must be identical.

No test address configured on interface interface_name disabling probe-based failure detection on it
Description:

In order for in.mpathd to perform probe-based failure detection on a NIC, it must be configured with a test address: IPv4, IPv6, or both.

The link has come up on interface_name more than 2 times in **the last minute; disabling repair until it stabilizes.**
the last minute; disabling failback until it stabilizes.
Description:

To limit the impact of interfaces with intermittent hardware (such as a bad cable), *in.mpathd* will not consider an IP interface with a frequently changing link state as repaired until the link state stabilizes.
In order to prevent interfaces with intermittent hardware, such as a bad cable, from causing repeated failovers and failbacks, in.mpathd does not failback to interfaces with frequently fluctuating link states.

Invalid failure detection time of *time*, assuming default of 10000 ms
Invalid failure detection time assuming default 10000
Description:

An invalid value was encountered for FAILURE_DETECTION_TIME in the /etc/default/mpathd file.

Too small failure detection time of *time*, assuming minimum of 100 ms
Too small failure detection time of time assuming minimum 100
Description:

251 SunOS 5.11 Last change: 8 Sep 2006 4

258 System Administration Commands in.mpathd(1M)

262 The minimum value that can be specified for
263 FAILURE_DETECTION_TIME is currently 100 milliseconds.

267 Invalid value for FAILBACK value
268 Description:

271 Valid values for the boolean variable FAILBACK are yes
272 or no.

276 Invalid value for TRACK_INTERFACES_ONLY_WITH_GROUPS value
277 Description:

280 Valid values for the boolean variable
281 TRACK_INTERFACES_ONLY_WITH_GROUPS are yes or no.

285 Cannot meet requested failure detection time of time ms on
286 (inet[6] interface_name) new failure detection time for
287 group group_name is time ms
288 Description:

291 The round trip time for ICMP probes is higher than
292 necessary to maintain the current failure detection
293 time. The network is probably congested or the probe
294 targets are loaded. in.mpathd automatically increases
295 the failure detection time to whatever it can achieve
296 under these conditions.

300 Improved failure detection time time ms on (inet[6]
301 interface_name) for group group_name
302 Description:

305 The round trip time for ICMP probes has now decreased
306 and in.mpathd has lowered the failure detection time
307 correspondingly.

311 **IP interface failure detected on interface_name**
318 *NIC failure detected on interface_name*
312 Description:

317 SunOS 5.11 Last change: 8 Sep 2006 5

324 System Administration Commands in.mpathd(1M)

328 **in.mpathd* has detected a failure on *interface_name*,**
329 **and has set the *IFF_FAILED* flag on *interface_name*,**
330 **ensuring that it will not be used for IP data traffic.**
335 *in.mpathd has detected NIC failure on interface_name,*
336 *and has set the IFF_FAILED flag on NIC interface_name.*

333 **IP interface repair detected on *interface_name***

340 *Successfully failed over from NIC interface_name1 to NIC*
341 *interface_name2*
334 Description:

337 **in.mpathd* has detected a repair on *interface_name*,**
338 **and has cleared the *IFF_FAILED* flag. Depending on the**
339 **administrative configuration, the *interface_name* may**
340 **again be used for IP data traffic.**
345 *in.mpathd has caused the network traffic to failover*
346 *from NIC interface_name1 to NIC interface_name2, which*
347 *is part of the multipathing group.*

351 *NIC repair detected on interface_name*
352 *Description:*

355 *in.mpathd has detected that NIC interface_name is*
356 *repaired and operational. If the IFF_FAILED flag on the*
357 *NIC was previously set, it will be reset.*

361 *Successfully failed back to NIC interface_name*
362 *Description:*

365 *in.mpathd has restored network traffic back to NIC*
366 *interface_name, which is now repaired and operational.*

343 The link has gone down on interface_name
344 Description:

347 **in.mpathd* has detected that the *IFF_RUNNING* flag for**
348 **interface_name* has been cleared, indicating the link**
349 **has gone down.**
374 *in.mpathd has detected that the IFF_RUNNING flag for NIC*
375 *interface_name has been cleared, indicating the link has*

gone down.

The link has come up on interface_name
Description:

***in.mpathd* has detected that the *IFF_RUNNING* flag for *interface_name* has been set, indicating the link has** *in.mpathd has detected that the IFF_RUNNING flag for NIC interface_name has been set, indicating the link has* come up.

```
**********************************************************
   16248 Fri Dec 19 04:18:56 2008
new/./ipmpstat.1m.txt
*** NO COMMENTS ***
**********************************************************
System Administration Commands                          ipmpstat(1M)

NAME
      ipmpstat - display IPMP subsystem status

SYNOPSIS

      ipmpstat [-n] [-o _field_ [-P]] -a|-g|-i|-p|-t

DESCRIPTION

      The *ipmpstat* command concisely displays information about
      the  IPMP subsystem.  It   supports five  different  output
      modes, each of which provides a different  view of the IPMP
      subsystem (address,  group, interface,  probe, and target),
      described below.  At most  one output mode may be specified
      per invocation, and the displayed information is guaranteed
      to be self-consistent.  It  also provides a parsable output
      format which may be used by scripts to examine the state of
      the IPMP subsystem.   Only  basic privileges are  needed to
      invoke  ipmpstat,  with the exception  of  probe mode which
      requires all privileges.

OPTIONS

      -a

           Display   IPMP   data  address information  ("address"
           output mode).

      -g

           Display IPMP group information ("group" output mode).

      -i

           Display IP  interface information  ("interface" output
           mode).

      -n

           Display   IP    addresses    numerically, rather  than
           attempting to  resolve them to hostnames.  This option
           may be used in any output mode.


      -o _field_[,...]

           Display  only  the specified  output fields, in order.
           The list   of field   names  is case-insensitive   and
           comma-separated.  The  field names  that are supported
           depend on  the selected output mode,  described below.
           The special field name "all"  may be used to print all
           fields for a given output mode.

      -p

           Display IPMP probe information ("probe" output mode).

      -t
```



```
           Display IPMP target information ("target" output mode).

      -P

           Display  using   a machine-parsable format,  described
           below.  If this option is  specified, an explicit list
           of fields must be specified using the *-o* option.


OUTPUT MODES

    Address Mode

        Address mode   displays   the state   of all   IPMP  data
        addresses on the system.  The following output fields are
        supported:

        ADDRESS     The  hostname (or IP address) associated with
                    the information.  Note that because duplicate
                    down addresses may exist, the address must be
                    taken  together   with  the GROUP to  form  a
                    unique identity.  For a  given IPMP group, if
                    duplicate addresses exist,  at most  one will
                    be displayed,  and an  up address will always
                    take precedence.

        STATE       The state of the address.  Either *up* if the
                    address  is    IFF_UP (see  ifconfig(1M)), or
                    *down* if the address is not IFF_UP.

        GROUP       The IPMP IP interface hosting the address.

        INBOUND     The underlying IP interface that will receive
                    packets for this address.  This may change in
                    response to  external   events such   as   IP
                    interface failure.  If  this  field is empty,
                    then the system   will not accept IP  packets
                    sent   to  this  address  (e.g.,  because the
                    address  is  down  or  because  there are  no
                    active IP interfaces left in the IPMP group).

        OUTBOUND    The underlying  IP interfaces  that will send
                    packets using this  source address.  This may
                    change in response to external events such as
                    IP interface   failure.    If  this  field is
                    empty, then the  system will not send packets
                    with this address as  a source (e.g., because
                    the address is  down or because there  are no
                    active IP interfaces left in the IPMP group).

        If *-o* is not specified, all output fields are displayed.

    Group Mode

        Group mode displays  the state of all  IPMP groups on the
        system.  The following output fields are supported:

        GROUP       The IPMP  IP interface  name associated  with
                    the  information.    For  the anonymous group
                    (see *in.mpathd(1M)*),   this  field will  be
                    empty.

        GROUPNAME   The  IPMP  group  name.   For   the anonymous
                    group, this field will be empty.

        STATE       The state of the group:
```

```
                    *ok*        All  interfaces in  the group are
                                usable.
                    *degraded*  Some (but not all)  interfaces in
                                the group are usable.
                    *failed*    No  interfaces  in the group are
                                usable.

        FDT         The probe-based failure   detection time.  If
                    probe-based  failure   detection is disabled,
                    this field will be empty.

        INTERFACES  The list  of underlying IP interfaces  in the
                    group.  The list is divided into three parts:

                    1. Active interfaces are listed first and not
                       enclosed   in any brackets or parenthesis.
                       Active interfaces are  those being used by
                       the system to send or receive data traffic.

                    2. *INACTIVE*  interfaces are listed next and
                       enclosed   in   parenthesis.    *INACTIVE*
                       interfaces are those that are functioning,
                       but   not    being   used  according    to
                       administrative policy.

                    3. Unusable  interfaces  are listed last  and
                       enclosed in brackets.  Unusable interfaces
                       are those that  cannot be  used at all  in
                       their    present     configuration  (e.g.,
                       *FAILED* or *OFFLINE*).

        If *-o* is not specified, all output fields are displayed.

    Interface Mode

        Interface mode displays   the state of all  IP interfaces
        that  are tracked by  *in.mpathd*  on the   system.   The
        following output fields are supported:

        INTERFACE   The  IP  interface  name associated  with the
                    information.

        ACTIVE      Either *yes* or *no*,  depending on if the IP
                    interface is being used   by the system   for
                    IP data traffic.

        GROUP       The IPMP IP  interface associated with the IP
                    interface.   For   IP   interfaces    in  the
                    anonymous group (see  *in.mpathd(1M)*),  this
                    field will be empty.

        FLAGS       Assorted information about the IP interface:

                    i  Unusable due to being *INACTIVE*.

                    s  Marked *STANDBY*.

                    m  Nominated  to send/receive IPv4  multicast
                       for its IPMP group.

                    b  Nominated  to receive  IPv4  broadcast for
                       its IPMP group.

                    M  Nominated  to send/receive IPv6  multicast
                       for its IPMP group.
```



```
                    d  Unusable due to being *down*.

                    h  Unusable due to being brought *OFFLINE* by
                       *in.mpathd*  because   of     a  duplicate
                       hardware address

        LINK        The state of link-based failure detection:

                    *up*        The link is up.

                    *down*      The link is down.

                    *unknown*   The   network   driver  does  not
                                detect link state changes.

        PROBE       The state of probe-based failure detection:

                    *ok*        Probes detect no problems.

                    *failed*    Probes detect failure.

                    *unknown*   Probes  cannot be  sent since no
                                suitable probe targets are known.

                    *disabled*  Probes have been disabled because
                                a unique IP  test address has not
                                been configured.

        STATE       The overall state of the interface:

                    *ok*        The   interface   is  online  and
                                functioning properly based on the
                                configured    failure   detection
                                methods.

                    *failed*    The interface is online but has a
                                link  state of *down*  or a probe
                                state of *failed*.

                    *offline*   The interface is offline.

                    *unknown*   The  interface is  online but may
                                or may not be functioning because
                                the  configured failure detection
                                methods are in *unknown* states.

        If *-o* is not specified, all output fields are displayed.

    Probe Mode

        Probe  mode displays information  about the  probes being
        sent by  *in.mpathd*.   Unlike other  output  modes, this
        mode runs  until explicitly terminated  using *^C*.   The
        following output fields are supported:

        TIME        The time the probe was sent, relative to when
                    *ipmpstat*   was started.  If   the probe was
                    sent prior to  starting *ipmpstat*,  the time
                    will be negative.

        PROBE       An identifier   representing  the probe.  The
                    identifier  will start   at  zero   and  will
                    monotonically increment  for each probe  sent
                    by *in.mpathd*   over a given  interface.  To
                    enable   more  detailed   analysis by  packet
```

monitoring tools, this identifier matches the *icmp_seq* field of the ICMP probe packet.

INTERFACE The IP interface the probe was sent on.

TARGET The hostname (or IP address) of the target the probe was sent to.

NETRTT The network round-trip-time for the probe. This is the time between when the IP module sends the probe and when the IP module receives the ack. If *in.mpathd* has concluded that the probe has been lost, this field will be empty.

RTT The total round-trip-time for the probe. This is the time between when *in.mpathd* starts executing the code to send the probe, and when it completes processing the ack. If *in.mpathd* has concluded that the probe has been lost, this field will be empty. Spikes in the total round-trip time that are not present in the network round-trip time indicate that the local system itself is overloaded.

RTTAVG The average round-trip-time to *TARGET* over *INTERFACE*. This aids identification of slow targets. If there is insufficient data to calculate the average, this field will be empty.

RTTDEV The standard deviation for the round-trip-time to *TARGET* over *INTERFACE*. This aids identification of jittery targets. If there is insufficient data to calculate the standard deviation, this field will be empty.

If *-o* is not specified, all fields except for *RTTAVG* and *RTTDEV* are displayed.

Target Mode

Target mode displays IPMP probe target information. The following output fields are supported:

INTERFACE The IP interface name associated with the information.

MODE The probe target discovery mode:

routes Probe targets found via the routing table.

multicast Probe targets found via multicast ICMP probes.

disabled Probe-based failure detection is disabled.

TESTADDR The hostname (or IP address) that will be used for sending and receiving probes. If a unique test address has not been configured, this field will be empty. Note that if an IP interface is configured with both IPv4 and

IPv6 test addresses, probe target information will be displayed separately for each test address.

TARGETS A space-separated list of probe target hostnames (or IP addresses), in firing order. If no probe targets could be found, this field will be empty.

If *-o* is not specified, all output fields are displayed.

OUTPUT FORMAT

By default, ipmpstat uses a human-friendly tabular format for its output modes, where each row contains one or more fields of information about a given object, which is in turn uniquely identified by one or more of those fields. In this format, a header identifying the fields is displayed above the table (and after each screenful of information), fields are separated by whitespace, empty fields are represented by *--*, and other visual aids are used.

Machine-parsable format also uses a tabular format, but is designed to be efficient to programmatically parse. Specifically, machine-parsable format differs from human-friendly format in the following ways:

- No headers are displayed.
- Fields with empty values yield no output, rather than *--*.
- Fields are separated by a single *:*, rather than variable amounts of whitespace.
- If multiple fields are requested, and a literal *:* or ** occur in a field's value, they are escaped by prefixing them with **.

EXAMPLES

Use the machine-parsable output format to create a *ksh* function that outputs the failure detection time of a given IPMP IP interface:

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

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/sbin

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Committed
Machine-Parsable Format	Committed

```
392     |_____________________________|_____________________________|
393     | Human-Friendly Format       | Not-an-Interface            |
394     |_____________________________|_____________________________|


397 SEE ALSO

399      in.mpathd(1M), ifconfig(1M), if_mpadm(1M)
```

****
20769 Fri Dec 19 04:18:57 2008
new/./route.1m.txt
Clearview IPMP manpages
****

System Administration Commands route(1M)

NAME

route - manually manipulate the routing tables

SYNOPSIS

```
route [-fnvq] sub-command [ [modifiers] args]

route [-fnvq] [-p [-R root-dir]] add | delete [modifiers] destination gatew
    [args]

route [-fnvq] change | get [modifiers] destination
    [gateway [args]]

route [-fn] monitor [modifiers]

route [-fnvq] flush [modifiers]

route -p [-R root-dir] show
```

DESCRIPTION

route manually manipulates the network routing tables. These tables are normally maintained by the system routing daemon, such as in.routed(1M) and in.ripngd(1M).

route supports a limited number of general options, but a rich command language. Users can specify an arbitrary request that can be delivered by means of the programmatic interface discussed in route(7P).

route uses a routing socket and the new message types RTM_ADD, RTM_DELETE, RTM_GET, and RTM_CHANGE. While only superusers can modify routing tables, the RTM_GET operation is allowed for non-privileged users.

OPTIONS

-f Flush the routing tables of all gateway entries. If you use the -f option in conjunction with any of the route sub-commands, route flushes the gateways before performing the sub-command. Specify the table to flush by placing the -inet or -inet6 modifier immediately after the -f option. If unspecified, flushing IPv4 (-inet) routes is the default.

-n Prevent attempts to print host and network names symbolically when reporting actions. This option is useful when name servers are unavailable.

-p Make changes to the network route tables persistent across system restarts. The operation is applied to the network routing tables first and, if successful, is then applied to the list of saved routes used at system startup. In determining whether an operation was successful, a failure to add a route that already exists or to delete a route that is not in the routing table is ignored. Particular care should be taken when using host or network names in persistent routes, as network-based name resolution services are not available at the time routes are added at startup.

-q Suppress all output.

-R root-dir Specify an alternate root directory where route applies changes. This option is ignored unless used in conjunction with the -p option. When -R is specified, route changes are applied only to the list of saved routes to be used at startup, not to the network routing tables. In addition, certain checks, **such as the existence of network interfaces** *such as the existance of network interfaces* used with -ifp, are skipped. This can be useful from within JumpStart scripts, where the root directory of the system being modified is in a location other than /.

-v Print additional details in verbose mode.

Subcommands

The following subcommands are supported:

add Add a route.

change Change aspects of a route (such as its gateway).

137 delete Delete a specific route.

140 flush Remove all gateway entries from the routing
141 table.

144 get Look up and display the route for a destination.

147 monitor Continuously report any changes to the routing
148 information base, routing lookup misses, or
149 suspected network partitionings.

152 show Display the list of routes to be applied at sys-
153 tem startup. Can be used only in conjunction with
154 the -p option.

158 The add and delete sub-commands have the following syntax:

160 route [-fnvq] cmd destination gateway [metric/netmask]

165 where cmd is add or delete, destination is the destination
166 host or network, and gateway is the next-hop intermediary
167 through which packets should be routed. Modifiers described
168 in OPERANDS can be placed anywhere on the command line.

171 The get and change sub-commands have the following syntax:

173 route [-fnvq] cmd destination [gateway [metric/netmask]]

178 where cmd is get or change, destination is the destination
179 host or network, and gateway is the next-hop intermediary
180 through which packets should be routed. Modifiers described
181 in OPERANDS can be placed anywhere on the command line.

184 The monitor sub-command has the following syntax:

186 route monitor [-inet | -inet6]

203 OPERANDS
204 route executes its sub-commands on routes to destinations by
205 way of gateways.

207 Destinations and Gateways
208 By default, destination and gateway addresses are inter-
209 preted as IPv4 addresses. All symbolic names are tried first
210 as a host name, using getipnodebyname(3SOCKET). If this
211 lookup fails in the AF_INET case, getnetbyname(3SOCKET)
212 interprets the name as that of a network.

215 Including an optional modifier on the command line before
216 the address changes how the route sub-command interprets it.

219 The following modifiers are supported:

221 -inet Force the address to be interpreted as an IPv4
222 address, that is, under the AF_INET address fam-
223 ily.

226 -inet6 Force the address to be interpreted as an IPv6
227 address, that is, under the AF_INET6 address fam-
228 ily.

232 For IPv4 addresses, routes to a particular host are by
233 default distinguished from those to a network by interpret-
234 ing the Internet address specified as the destination. If
235 the destination has a local address part (that is, the por-
236 tion not covered by the netmask) of 0, or if the destination
237 is resolved as the symbolic name of a network, then the
238 route is assumed to be to a network; otherwise, it is
239 presumed to be a route to a host.

242 You can force this selection by using one of the following
243 modifiers:

245 -host Force the destination to be interpreted as a host.

248 -net Force the destination to be interpreted as a net-
249 work.

253 For example:

Destination	Destination Equivalent
128.32	-host 128.0.0.32
128.32.130	-host 128.32.0.130
-net 128.32	128.32.0.0
-net 128.32.130	128.32.130.0

Two modifiers avoid confusion between addresses and keywords (for example., host used as a symbolic host name). You can distinguish a destination by preceding it with the -dst modifier. You can distinguish a gateway address by using the -gateway modifier. If the destination is directly reachable by way of an interface requiring no intermediary IP router to act as a gateway, this can be indicated by using the -interface or -iface modifier.

In the following example, the route does not refer to an external gateway (router), but rather to one of the machine's interfaces. Packets with IP destination addresses matching the destination and mask on such a route are sent out on the interface identified by the gateway address. For interfaces using the ARP protocol, this type of route is used to specify that all matching destinations are local to the physical link. That is, a host could be configured to ARP for all addresses, without regard to the configured interface netmask, by adding a default route using this command. For example:

```
example# route add default hostname -interface
```

where gateway address hostname is the name or IP address associated with the network interface over which all matching packets should be sent. On a host with a single network interface, hostname is usually the same as the nodename returned by the uname -n command. See uname(1).

For backward compatibility with older systems, directly reachable routes can also be specified by placing a 0 after the gateway address:

```
example# route add default hostname 0
```


This value was once a route metric, but this metric is no longer used. If the value is specified as 0, then the destination is directly reachable (equivalent to specifying -interface). If it is non-zero but cannot be interpreted as a subnet mask, then a gateway is used (default).

With the AF_INET address family or an IPv4 address, a separate subnet mask can be specified. This can be specified in one of the following ways:

- IP address following the gateway address . This is typically specified in decimal dot notation as for **inet_addr(3SOCKET) rather than in symbolic form.** *inet_addr(3SOCKET) rather than in symbollic form.*
- IP address following the -netmask qualifier.
- Slash character and a decimal length appended to the destination address.

If a subnet mask is not specified, the mask used is the subnet mask of the output interface selected by the gateway address, if the classful network of the destination is the same as the classful network of the interface. Otherwise, the classful network mask for the destination address is used.

Each of the following examples creates an IPv4 route to the destination 192.0.2.32 subnet with a subnet mask of 255.255.255.224:

```
example# route add 192.0.2.32/27 somegateway
example# route add 192.0.2.32 -netmask 255.255.255.224 somegateway
example# route add 192.0.2.32 somegateway 255.255.255.224
```

For IPv6, only the slash format is accepted. The following example creates an IPv6 route to the destination 33fe:: with a netmask of 16 one-bits followed by 112 zero-bits.

```
example# route add -inet6 3ffe::/16 somegateway
```

In cases where the gateway does not uniquely identify the output interface (for example, when several interfaces have the same address), you can use the -ifp ifname modifier to

specify the interface by name. For example, *-ifp lo0* associates the route with the *lo0* interface. If the named interface is an underlying interface in an IPMP group, then requests to add a route will automatically be translated to the corresponding IPMP IP interface, and requests to delete or change a route on an underlying interface will fail.
specify the interface by name. For example, -ifp lo0 associates the route with the lo0 interface.

Routing Flags

Routes have associated flags that influence operation of the protocols when sending to destinations matched by the routes. These flags can be set (and in some cases cleared, indicated by ~) by including the following modifiers on the command line:

Modifier	Flag	Description
-interface	~RTF_GATEWAY	Destination is directly reachable
-iface	~RTF_GATEWAY	Alias for interface modifier
-static	RTF_STATIC	Manually added route
-nostatic	~RTF_STATIC	Pretend route was added by kernel or routing daemon
-reject	RTF_REJECT	Emit an ICMP unreachable when matched
-blackhole	RTF_BLACKHOLE	Silently discard packets duing updates
-proto1	RTF_PROTO1	Set protocol specific routing flag #1
-proto2	RTF_PROTO2	Set protocol specific routing flag #2
-private	RTF_PRIVATE	Do not advertise this route
-multirt	RTF_MULTIRT	Creates the specified redundant route
-setsrc	RTF_SETSRC	Assigns the default source address

The optional modifiers -rtt, -rttvar, -sendpipe, -recvpipe, -mtu, -hopcount, -expire, and -ssthresh provide initial values to quantities maintained in the routing entry by transport level protocols, such as TCP. These can be individually locked either by preceding each modifier to be locked by the -lock meta-modifier, or by specifying that all ensuing metrics can be locked by the -lockrest meta-modifier.

Some transport layer protocols can support only some of these metrics. The following optional modifiers are supported:

-expire Lifetime for the entry. This optional modifier is not currently supported.

-hopcount Maximum hop count. This optional modifier is not currently supported.

-mtu Maximum MTU in bytes.

-recvpipe Receive pipe size in bytes.

-rtt Round trip time in microseconds.

-rttvar Round trip time variance in microseconds.

-sendpipe Send pipe size in bytes.

-ssthresh Send pipe size threshold in bytes.

-secattr Security attributes of the route. This modifier is available only if the system is configured with the Solaris Trusted Extensions feature.

The -secattr modifier has the following format:

```
min_sl=val,max_sl=val,doi=val,cipso
```

or:

```
sl=VAL,doi=VAL,cipso
```

In the first form, above, the val for min_sl and max_sl is a sensitivity label in either hex or string form. The val for doi is a non-negative integer. The route will apply only for packets with the same domain of interpretation as defined by the doi value and within the accreditation range defined by the min_sl and max_sl values. The cipso keyword is optional and set by default. Valid min_sl, max_sl and doi keyword/value pairs are mandatory. Note that if val contains a space, it must be protected by double quotes.

The second form, above, is equivalent to specifying the first form with the same VAL for min_sl and max_sl. The second form should be used for the get command, because get uses only a single sensitivity label.

Compatibility

The modifiers host and net are taken to be equivalent to -host and -net. To specify a symbolic address that matches one of these names, use the dst or gateway keyword to distinguish it. For example: -dst host

The following two flags are also accepted for compatibility with older systems, but have no effect.

Modifier	Flag
-cloning	RTF_CLONING
-xresolve	RTF_XRESOLVE

The -ifa hostname modifier is also accepted, but has no effect.

FILES

/etc/defaultrouter List of default routers

/etc/hosts List of host names and net addresses

/etc/networks List of network names and addresses

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

SEE ALSO

uname(1), in.ripngd(1M), in.routed(1M), netstat(1M), routed(1M), ioctl(2), getipnodebyname(3SOCKET), getnetbyname(3SOCKET), inet_addr(3SOCKET), defaultrouter(4), hosts(4), networks(4), attributes(5), ARP(7P), ip(7P), route(7P), routing(7P)

DIAGNOSTICS

add [host| network] destination:gateway flags

The specified route is being added to the tables. The values printed are from the routing table entry supplied in the ioctl(2) call. If the gateway address used was not the primary address of the gateway (the first one

returned by getipnodebyname(3SOCKET)) the gateway address is printed numerically as well as symbolically.

delete [host| network] destination:gateway flags
change [host| network] destination:gateway flags

As add, but when deleting or changing an entry.

destination done

When the -f flag is specified, or the flush sub-command is used, each routing table entry deleted is indicated with a message of this form.

Network is unreachable

An attempt to add a route failed because the gateway listed was not on a directly-connected network. Give the next-hop gateway instead.

not in table

A delete operation was attempted for an entry that is not in the table.

entry exists

An add operation was attempted for a route that already exists in the kernel.

routing table overflow

An operation was attempted, but the system was unable to allocate memory to create the new entry.

insufficient privileges

An attempt to add, delete, change, or flush a route failed because the calling process does not have appropriate privileges.

NOTES
Specifying that destinations are local (with the -interfacemodifier) assumes that the routers implement proxy ARP,

meaning that they respond to ARP queries for all reachable destinations. Normally, using either router discovery or RIP is more reliable and scalable than using proxy ARP. See in.routed(1M) for information related to RIP.

Combining the all destinations are local route with subnet or network routes can lead to unpredictable results. The search order as it relates to the all destinations are local route are undefined and can vary from release to release.

****
14100 Fri Dec 19 04:18:58 2008
new/./route.7p.txt
Clearview IPMP manpages
****

5 NAME
6 route - kernel packet forwarding database

8 SYNOPSIS
9 #include <sys/types.h>
10 #include <sys/socket.h>
11 #include <net/if.h>
12 #include <net/route.h>

14 int socket(PF_ROUTE, SOCK_RAW, int protocol);

17 DESCRIPTION
18 UNIX provides some packet routing facilities. The kernel
19 maintains a routing information database, which is used in
20 selecting the appropriate network interface when transmit-
21 ting packets.

24 A user process (or possibly multiple co-operating processes)
25 maintains this database by sending messages over a special
26 kind of socket. This supplants fixed size ioctl(2)'s speci-
27 fied in routing(7P). Routing table changes may only be car-
28 ried out by the superuser.

31 The operating system may spontaneously emit routing messages
32 in response to external events, such as receipt of a re-
33 direct, or failure to locate a suitable route for a request.
34 The message types are described in greater detail below.

37 Routing database entries come in two flavors: entries for a
38 specific host, or entries for all hosts on a generic subnet-
39 work (as specified by a bit mask and value under the mask).
40 The effect of wildcard or default route may be achieved by
41 using a mask of all zeros, and there may be hierarchical
42 routes.

45 When the system is booted and addresses are assigned to the
46 network interfaces, the internet protocol family installs a
47 routing table entry for each interface when it is ready for
48 traffic. Normally the protocol specifies the route through
49 each interface as a direct connection to the destination
50 host or network. If the route is direct, the transport
51 layer of a protocol family usually requests the packet be
52 sent to the same host specified in the packet. Otherwise,
53 the interface is requested to address the packet to the
54 gateway listed in the routing entry, that is, the packet is
55 forwarded.

71 When routing a packet, the kernel attempts to find the most
72 specific route matching the destination. If no entry is
73 found, the destination is declared to be unreachable, and a
74 routing-miss message is generated if there are any listeners
75 on the routing control socket (described below). If there
76 are two different mask and value-under-the-mask pairs that
77 match, the more specific is the one with more bits in the
78 mask. A route to a host is regarded as being supplied with a
79 mask of as many ones as there are bits in the destination.

82 A wildcard routing entry is specified with a zero destina-
83 tion address value, and a mask of all zeroes. Wildcard
84 routes are used when the system fails to find other routes
85 matching the destination. The combination of wildcard routes
86 and routing redirects can provide an economical mechanism
87 for routing traffic.

90 One opens the channel for passing routing control messages
91 by using the socket call shown in the section above. There
92 can be more than one routing socket open per system.

95 Messages are formed by a header followed by a small number
96 of sockaddrs, whose length depend on the address family.
97 sockaddrs are interpreted by position. An example of a type
98 of message with three addresses might be a CIDR prefix
99 route: Destination, Netmask, and Gateway. The interpretation
100 of which addresses are present is given by a bit mask within
101 the header, and the sequence is least significant to most
102 significant bit within the vector.

105 Any messages sent to the kernel are returned, and copies are
106 sent to all interested listeners. The kernel provides the
107 process ID of the sender, and the sender may use an addi-
108 tional sequence field to distinguish between outstanding
109 messages. However, message replies may be lost when kernel
110 buffers are exhausted.

113 The protocol parameter specifies which messages an applica-
114 tion listening on the routing socket is interested in see-
115 ing, based on the the address family of the sockaddrs
116 present. Currently, you can specify AF_INET and AF_INET6 to
117 filter the messages seen by the listener, or alternatively,
118 you can specify AF_UNSPEC to indicate that the listener is
119 interested in all routing messages.

Protocols route(7P)

The kernel may reject certain messages, and will indicate this by filling in the rtm_errno field of the rt_msghdr struct (see below). The following codes may be returned:

EEXIST If requested to duplicate an existing entry

ESRCH If requested to delete a non-existent entry

ENOBUFS If insufficient resources were available to install a new route.

EPERM If the calling process does not have appropriate privileges to alter the routing table.

In the current implementation, all routing processes run locally, and the values for rtm_errno are available through the normal errno mechanism, even if the routing reply message is lost.

A process may avoid the expense of reading replies to its own messages by issuing a setsockopt(3SOCKET) call indicating that the SO_USELOOPBACK option at the SOL_SOCKET level is to be turned off. A process may ignore all messages from the routing socket by doing a shutdown(3SOCKET) system call for further input.

By default, underlying IP interfaces in an IPMP group are not visible to routing sockets. As such, routing sockets will not receive events related to underlying IP interfaces in an IPMP group. For consistency, when an IP interface is placed into an IPMP group, *RTM_DELADDR* messages will be generated for each of the *IFF_UP* addresses that are not migrated to the corresponding IPMP IP interface, and an *RTM_IFINFO* message will be sent indicating that the interface is now down. Similarly, when an underlying interface is removed from an IPMP group, an *RTM_IFINFO* message will be sent indicating that the interface is again up, and *RTM_NEWADDR* messages will be generated for each *IFF_UP* address found on the interface.

The *RT_AWARE* socket option at the *SOL_ROUTE* level allows an application to indicate its awareness of certain features, which controls routing socket behavior. The supported values are:

***RTAW_DEFAULT* Default awareness.**

***RTAW_UNDER_IPMP* IPMP underlying interface awareness. When this is enabled, underlying IP interfaces in an IPMP group will remain visible to the routing socket and events related to them will continue to be generated.**

An *RTM_ADD* request tied to an underlying IP interface in an IPMP group will be translated to an *RTM_ADD* request for its corresponding IPMP IP interface. All routing socket requests other than *RTM_ADD* and *RTM_GET* will fail when issued on an underlying IP interface in an IPMP group.

If a route is in use when it is deleted, the routing entry is marked down and removed from the routing table, but the resources associated with it are not reclaimed until all references to it are released.

The RTM_IFINFO, RTM_NEWADDR, and RTM_ADD messages associated with interface configuration (setting the IFF_UP bit) are normally delayed until after Duplicate Address Detection completes. Thus, applications that configure interfaces and wish to wait until the interface is ready can wait until RTM_IFINFO is returned and SIOCGLIFFLAGS shows that IFF_DUPLICATE is not set.

Messages

User processes can obtain information about the routing entry to a specific destination by using a RTM_GET message.

Messages include:

```
#define RTM_ADD      0x1   /* Add Route */
#define RTM_DELETE   0x2   /* Delete Route */
#define RTM_CHANGE   0x3   /* Change Metrics, Flags, or Gateway */
#define RTM_GET      0x4   /* Report Information */
#define RTM_LOSING   0x5   /* Kernel Suspects Partitioning */
#define RTM_REDIRECT 0x6   /* Told to use different route */
#define RTM_MISS     0x7   /* Lookup failed on this address */
#define RTM_LOCK     0x8   /* fix specified metrics */
#define RTM_OLDADD   0x9   /* caused by SIOCADDRT */
#define RTM_OLDDEL   0xa   /* caused by SIOCDELRT */
#define RTM_RESOLVE  0xb   /* request to resolve dst to LL addr */
#define RTM_NEWADDR  0xc   /* address being added to iface */
#define RTM_DELADDR  0xd   /* address being removed from iface */
#define RTM_IFINFO   0xe   /* iface going up/down etc. */
```

A message header consists of:

```
261        struct rt_msghdr {
262         ushort_t rtm_msglen;    /* to skip over non-understood messages */
263         uchar_t  rtm_version;   /* future binary compatibility */
264         uchar_t  rtm_type;      /* message type */
265         ushort_t rtm_index;     /* index for associated ifp */
266         pid_t   rtm_pid;        /* identify sender */
267         int     rtm_addrs;      /* bitmask identifying sockaddrs in msg */
268         int     rtm_seq;        /* for sender to identify action */
269         int     rtm_errno;      /* why failed */
270         int     rtm_flags;      /*  flags,  incl  kern  &  message, e.g., DONE *
271         int     rtm_use;        /* from rtentry */
272         uint_t  rtm_inits;      /* which values we are initializing */

274        struct  rt_metrics rtm_rmx;   /* metrics themselves */
275            };



279      where

281        struct rt_metrics {
282         uint32_t rmx_locks;      /* Kernel must leave  these  values alone */
283         uint32_t rmx_mtu;        /* MTU for this path */
284         uint32_t rmx_hopcount;   /* max hops expected */
285         uint32_t rmx_expire;     /* lifetime for route, e.g., redirect */
286         uint32_t rmx_recvpipe;   /* inbound delay-bandwidth  product */
287         uint32_t rmx_sendpipe;   /* outbound delay-bandwidth product */
288         uint32_t rmx_ssthresh;   /* outbound gateway buffer limit */
289         uint32_t rmx_rtt;        /* estimated round trip time */
290         uint32_t rmx_rttvar;     /* estimated rtt variance */
291         uint32_t rmx_pksent;     /* packets sent using this route */


295 SunOS 5.11          Last change: 25 July 2006                  4




302 Protocols                                          route(7P)


306        };

308        /* Flags include the values */


311        #define RTF_UP          0x1     /* route usable */
312        #define RTF_GATEWAY     0x2     /* destination is a gateway */
313        #define RTF_HOST        0x4     /* host entry (net otherwise) */
314        #define RTF_REJECT      0x8     /* host or net unreachable */
315        #define RTF_DYNAMIC     0x10    /* created dynamically(by redirect) */
316        #define RTF_MODIFIED    0x20    /* modified dynamically(by redirect) */
317        #define RTF_DONE        0x40    /* message confirmed */
318        #define RTF_MASK        0x80    /* subnet mask present */
319        #define RTF_CLONING     0x100   /* generate new routes on use */
320        #define RTF_XRESOLVE    0x200   /* external daemon resolves name */
321        #define RTF_LLINFO      0x400   /* generated by ARP */
322        #define RTF_STATIC      0x800   /* manually added */
323        #define RTF_BLACKHOLE   0x1000  /* just discard pkts (during updates) */
324        #define RTF_PRIVATE     0x2000  /* do not advertise this route */
325        #define RTF_PROTO2      0x4000  /* protocol specific routing flag #2 */
```



```
326        #define RTF_PROTO1      0x8000  /* protocol specific routing flag #1 */

328        /* Specifiers for metric values in rmx_locks and rtm_inits are */

330        #define RTV_MTU         0x1     /* init or lock _mtu */
331        #define RTV_HOPCOUNT    0x2     /* init or lock _hopcount */
332        #define RTV_EXPIRE      0x4     /* init or lock _expire */
333        #define RTV_RPIPE       0x8     /* init or lock _recvpipe */
334        #define RTV_SPIPE       0x10    /* init or lock _sendpipe */
335        #define RTV_SSTHRESH    0x20    /* init or lock _ssthresh */
336        #define RTV_RTT         0x40    /* init or lock _rtt */
337        #define RTV_RTTVAR      0x80    /* init or lock _rttvar */

339        /* Specifiers for which addresses are present in  the  messages are */

341        #define RTA_DST         0x1     /* destination sockaddr present */
342        #define RTA_GATEWAY     0x2     /* gateway sockaddr present */
343        #define RTA_NETMASK     0x4     /* netmask sockaddr present */
344        #define RTA_GENMASK     0x8     /* cloning mask sockaddr present */
345        #define RTA_IFP         0x10    /* interface name sockaddr present */
346        #define RTA_IFA         0x20    /* interface addr sockaddr present */
347        #define RTA_AUTHOR      0x40    /* sockaddr for author of redirect */
348        #define RTA_BRD         0x80    /* for NEWADDR, broadcast or p-p dest addr


351 SEE ALSO
352      ioctl(2),      setsockopt(3SOCKET),       shutdown(3SOCKET),
353      routing(7P)

355 NOTES
356      Some of the metrics may not be implemented and return  zero.
357      The implemented metrics are set in rtm_inits.


361 SunOS 5.11          Last change: 25 July 2006                  5
```

```
**********************************************************
   12781 Fri Dec 19 04:18:59 2008
new/./socket.h.txt
Clearview IPMP manpages
**********************************************************
```

```
Headers                                              socket.h(3HEAD)



NAME
     socket.h, socket - Internet Protocol family

SYNOPSIS
     #include <sys/socket.h>


DESCRIPTION
     The <sys/socket.h> header defines the unsigned integral type
     sa_family_t through typedef.


     The <sys/socket.h> header  defines  the  sockaddr  structure
     that includes the following members:

       sa_family_t   sa_family     /* address family */
       char          sa_data[]     /* socket   address (variable-length
                                      data) */



  libxnet Interfaces
     The <sys/socket.h> header defines the msghdr  structure  for
     libxnet interfaces that includes the following members:

       void            *msg_name         /* optional address */
       socklen_t       msg_namelen       /* size of address */
       struct iovec    *msg_iov          /* scatter/gather array */
       int             msg_iovlen        /* members in msg_iov */
       void            *msg_control      /* ancillary data, see below */
       socklen_t       msg_controllen    /* ancillary data buffer len */
       int             msg_flags         /* flags on received message */



     The <sys/socket.h> header defines the cmsghdr structure  for
     libxnet that includes the following members:

       socklen_t  cmsg_len      /* data byte count, including hdr */
       int        cmsg_level    /* originating protocol */
       int        cmsg_type     /* protocol-specific type */



     Ancillary data consists of a sequence of  pairs,  each  con-
     sisting of a cmsghdr structure followed by a data array. The
     data array contains the  ancillary  data  message,  and  the
     cmsghdr  structure  contains  descriptive  information  that
     allows an application to correctly parse the data.




SunOS 5.11          Last change: 03 Aug 2006                  1
```



```
Headers                                              socket.h(3HEAD)



     The values for cmsg_level will be legal values for the level
     argument to the getsockopt() and setsockopt() functions. The
     SCM_RIGHTS type is supported for level SOL_SOCKET.


     Ancillary data is also possible at  the  socket  level.  The
     <sys/socket.h>  header  defines the following macros for use
     as the cmsg_type values when cmsg_level is SOL_SOCKET.

     SCM_RIGHTS    Indicates that the  data  array  contains  the
                   access rights to be sent or received.


     SCM_UCRED     Indicates  that  the  data  array  contains  a
                   ucred_t  to  be  received.  The ucred_t is the
                   credential of the sending process at the  time
                   the  message was sent. This is a Sun-specific,
                   Evolving interface. See ucred_get(3C).


     The IPv4 data formats generally use the same values for data
     passed  back  in  cmsghdr  as for setsockopt() to enable the
     feature. The IPv4 data formats are  listed  below  with  the
     associated payload for each.
     IPPROTO_IP
     IP_RECVDSTADDR

         ipaddr_t, IP address


     IPPROTO_IP
     IP_RECVOPTS

         variable-length IP options, up to 40 bytes


     IPPROTO_IP
     IP_RECVIF

         uint_t, ifIndex number


     IPPROTO_IP
     IP_RECVSLLA

         struct sockaddr_dl, link layer address


     IPPROTO_IP
     IP_RECVTTL



SunOS 5.11          Last change: 03 Aug 2006                  2
```

137 uint8_t

140 SOL_SOCKET
141 SO_RECVUCRED

143 ucred_t - cmsghdr.cmsg_type is SCM_UCRED, not
144 SO_RECVUCRED

148 The IPv6 data formats use different values for enabling the
149 option and for passing the value back to the application.
150 **The IPv6 data formats are listed below with the associated**
150 *The IPv6 data formats are listed below with the associated*
151 payload for each.

153 IPPROTO_IPV6
154 IPV6_RECVPKTINFO

156 in_pktinfo, cmsg_type IPV6_PKTINFO

159 IPPROTO_IPV6
160 IPV6_RECVTCLASS

162 uint_t, cmsg_type IPV6_TCLASS

165 IPPROTO_IPV6
166 IPV6_RECVPATHMTU

168 ip6_mtuinfo, cmsg_type IPV6_PATHMTU

171 IPPROTO_IPV6
172 IPV6_RECVHOPLIMIT

174 uint_t, cmsg_type IPV6_HOPLIMIT

177 IPPROTO_IPV6
178 IPV6_RECVHOPOPTS

180 variable-length IPv6 options, cmsg_type IPV6_HOPOPTS

183 IPPROTO_IPV6
184 IPV6_RECVDSTOPTS

186 variable-length IPv6 options, cmsg_type IPV6_DSTOPTS

203 IPPROTO_IPV6
204 IPV6_RECVRTHDR

206 variable-length IPv6 options, cmsg_type IPV6_RTHDR

209 IPPROTO_IPV6
210 IPV6_RECVRTHDRDSTOPTS

212 variable-length IPv6 options, cmsg_type IPV6_DSTOPTS

216 The <sys/socket.h> header defines the following macros to
217 gain access to the data arrays in the ancillary data associ-
218 ated with a message header:

220 CMSG_DATA(cmsg)

222 If the argument is a pointer to a cmsghdr structure,
223 this macro returns an unsigned character pointer to the
224 data array associated with the cmsghdr structure.

227 CMSG_NXTHDR(mhdr, cmsg)

229 If the first argument is a pointer to a msghdr structure
230 and the second argument is a pointer to a cmsghdr struc-
231 ture in the ancillary data, pointed to by the
232 msg_control field of that msghdr structure, this macro
233 returns a pointer to the next cmsghdr structure, or a
234 null pointer if this structure is the last cmsghdr in
235 the ancillary data.

238 CMSG_FIRSTHDR(mhdr)

240 If the argument is a pointer to a msghdr structure, this
241 macro returns a pointer to the first cmsghdr structure
242 in the ancillary data associated with this msghdr struc-
243 ture, or a null pointer if there is no ancillary data
244 associated with the msghdr structure.

247 CMSG_SPACE(len)

249 Given the length of an ancillary data object,
250 CMSG_SPACE() returns the space required by the object
251 and its cmsghdr structure, including any padding needed
252 to satisfy alignment requirements. This macro can be
253 used, for example, to allocate space dynamically for the
254 ancillary data. This macro should not be used to

initialize the cmsg_len member of a cmsghdr structure. Use the CMSG_LEN() macro instead.

CMSG_LEN(len)

Given the length of an ancillary data object, CMSG_LEN() returns the value to store in the cmsg_len member of the cmsghdr structure, taking into account any padding needed to satisfy alignment requirements.

The <sys/socket.h> header defines the linger structure that includes the following members:

```
int   l_onoff   /* indicates whether linger option is enabled */
int   l_linger  /* linger time, in seconds */
```

The <sys/socket.h> header defines the following macros:

SOCK_DGRAM	Datagram socket
SOCK_STREAM	Byte-stream socket
SOCK_SEQPACKET	Sequenced-packet socket

The *<sys/socket.h>* header defines the following macros for use as the *level* argument of *setsockopt()* and getsockopt()*.
The <sys/socket.h> header defines the following macro for use as the level argument of setsockopt() and getsockopt().

SOL_SOCKET	**Options to be accessed at the socket level, not the protocol level.**
SOL_SOCKET	*Options to be accessed at socket level, not protocol level.*
SOL_ROUTE	**Options to be accessed at the routing socket level, not the protocol level.**

The <sys/socket.h> header defines the following macros for use as the option_name argument in getsockopt() or setsockopt() calls:

SO_DEBUG	Debugging information is being recorded.

SO_ACCEPTCONN	Socket is accepting connections.

SO_BROADCAST	Transmission of broadcast messages is supported.
SO_REUSEADDR	Reuse of local addresses is supported.
SO_KEEPALIVE	Connections are kept alive with periodic messages.
SO_LINGER	Socket lingers on close.
SO_OOBINLINE	Out-of-band data is transmitted in line.
SO_SNDBUF	Send buffer size.
SO_RCVBUF	Receive buffer size.
SO_ERROR	Socket error status.
SO_TYPE	Socket type.
SO_RECVUCRED	Request the reception of user credential ancillary data. This is a Sun-specific, Evolving interface. See ucred_get(3C).
SO_MAC_EXEMPT	Mandatory Access Control (MAC) exemption for unlabeled peers. This option is available only if the system is configured with Trusted Extensions.
SO_ALLZONES	Bypass zone boundaries (privileged).

The <sys/socket.h> header defines the following macros for use as the valid values for the msg_flags field in the msghdr structure, or the flags parameter in recvfrom(), recvmsg(), sendto(), or sendmsg() calls:

MSG_CTRUNC Control data truncated.

MSG_EOR Terminates a record (if supported by the protocol).

MSG_OOB Out-of-band data.

MSG_PEEK Leave received data in queue.

MSG_TRUNC Normal data truncated.

MSG_WAITALL Wait for complete message.

The <sys/socket.h> header defines the following macros:

AF_UNIX UNIX domain sockets

AF_INET Internet domain sockets

The <sys/socket.h> header defines the following macros:

SHUT_RD Disables further receive operations.

SHUT_WR Disables further send operations.

SHUT_RDWR Disables further send and receive operations.

libsocket Interfaces

The <sys/socket.h> header defines the msghdr structure for libsocket interfaces that includes the following members:

```
void           *msg_name        /* optional address */
socklen_t      msg_namelen      /* size of address */
struct iovec   *msg_iov         /* scatter/gather array */
int            msg_iovlen       /* # elements in msg_iov */
caddr_t        msg_accrights    /* access rights sent/received */
```


The msg_name and msg_namelen parameters specify the destination address when the socket is unconnected The msg_name can be specified as a NULL pointer if no names are desired or

required. The msg_iov and msg_iovlen parameters describe the scatter-gather locations, as described in read(2). The msg_accrights parameter specifies the buffer in which access rights sent along with the message are received. The msg_accrightslen specifies the length of the buffer.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Standard

SEE ALSO

accept(3SOCKET), accept(3XNET), bind(3SOCKET), bind(3XNET), connect(3SOCKET), connect(3XNET), getpeername(3SOCKET), getpeername(3XNET), getpeerucred(3C), getsockname(3SOCKET), getsockname(3XNET), getsockopt(3SOCKET), getsockopt(3XNET), libsocket(3LIB), listen(3SOCKET), listen(3XNET), recv(3SOCKET), recv(3XNET), recvfrom(3SOCKET), recvfrom(3XNET), recvmsg(3SOCKET), recvmsg(3XNET), send(3SOCKET), send(3XNET), sendmsg(3SOCKET), sendmsg(3XNET), sendto(3SOCKET), sendto(3XNET), setsockopt(3SOCKET), setsockopt(3XNET), shutdown(3SOCKET), shutdown(3XNET), socket(3SOCKET), socket(3XNET), socketpair(3SOCKET) socketpair(3XNET), ucred_get(3C)attributes(5), standards(5)