Solaris 8 Administrator's Guide

By Paul Watters
January 2002
0-596-00073-1, Order Number: 0731
400 pages, $39.95 US $59.95 CA #28.50 UK

Chapter 4
Network Configuration

After undertaking the complex tasks required to configure a single host, planning and setting up an entire network can be daunting. In this chapter, you'll learn how to configure a Solaris-based network, including the configuration of single or multiple network interfaces, static and dynamic routing, and network troubleshooting. In addition, examples for enabling devices and testing interfaces will be provided.

Creating Networks and Subnets

While Solaris systems are capable of operating in an isolated, non-networked environment, Solaris is a strongly network-oriented operating system. It provides the following tools to support networking, both between hosts on a local area network and to the worldwide Internet:

- Support for single, dual, and quad ethernet devices
- Standardized network device naming
- Support for a wide variety of network devices
- Configuration of interfaces to support IPv4 and IPv6
- Routing using static and dynamic protocols
- Troubleshooting and performance measurement
- Blocking/acess filtering on all TCP and UDP ports
- Transmission using Ethernet and FDDI
- Support for Asynchronous Transfer Mode (ATM) networks

In combination, these features make it easy to construct Solaris networks, especially networks in which Solaris systems are assigned backbone functions in routing and packet filtering.

A typical Solaris local area network will contain one or more servers, which provide network services to local clients. These clients can be other Solaris systems, but are just as likely to be Linux, Microsoft Windows, or other Unix systems. In some network designs, each major service is located on its own system, to prevent downtime on one system from disabling access to all services. This brand of server role diversification is taken one step further by the E10000 system, which can be logically partitioned to form 64 independent virtual servers, all physically located on the same machine. Thus, if one domain is taken offline for service, other domains are unaffected.

The numbers and types of services provided on a Solaris local area network are virtually endless, but a typical configuration would include the following service types:

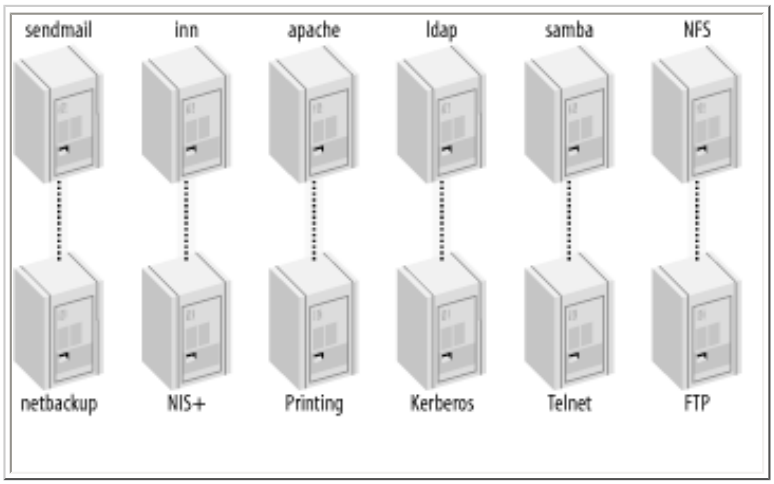
Mail server	Authentication server
USENET server	Resource management server
UNIX-compatible file server	Remote access server
PC-compatible file server	Remote procedure call server
Backup server	WWW server
Print server	Directory server

Solaris provides the following services that implement these service types:

Sendmail	Kerberos
Inn	NIS+
NFS	Telnet and FTP
Samba	RPC daemon
Netbackup	Apache
System V and BSD print systems	LDAP

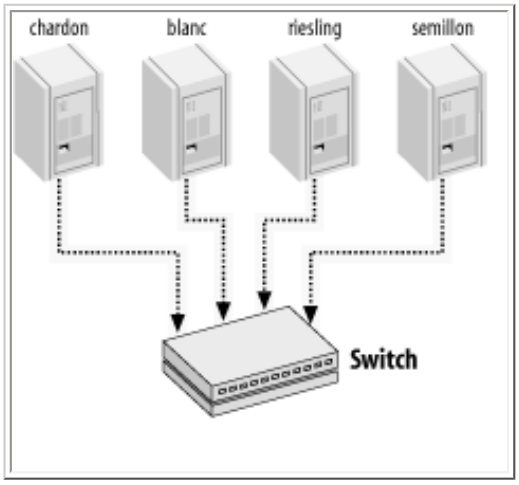
Figure 4-1 shows a sample server setup for a single Class C network.

Figure 4-1.Sample Solaris server configuration for a single Class C network



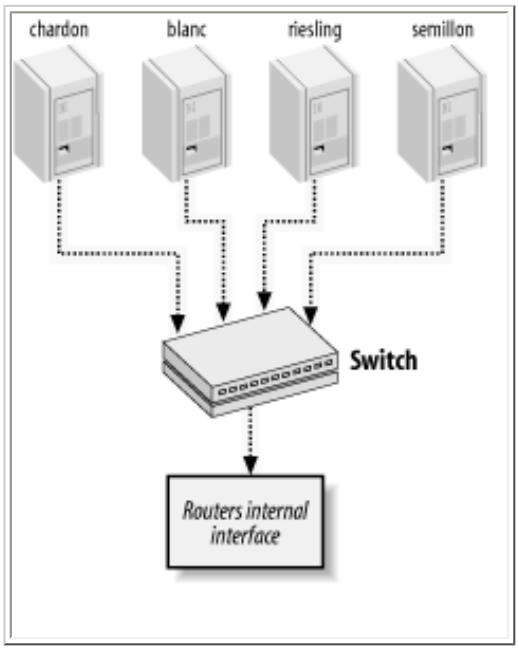
Once a server setup has been decided for the local area network, a number of other issues, such as assigning IP address ranges to individual subnets and IP addresses to individual hosts, must be addressed. (Details of how to assign these addresses are provided in Chapter 2.) A modern network is generally connected using 10/100M Ethernet cabling, where hosts on the same subnet are ultimately connected to a single router via a switch or a hub. Figure 4-2 shows a single local network, with the hosts *chardon*, *blanc*, *riesling*, and *semillon* connected via a central switch. If all hosts are cabled with 100M ethernet cable, all traffic on the network is transmitted at the 100M rate. Mixed mode cabling and packet transmission rates can be problematic and, since most Solaris network interface cards now support 100M, standardizing on this rate is preferable. This simple network has no gateway, does not connect to other networks or to the Internet, and does not require a router.

Figure 4-2.Simple Solaris network for a single Class C network not connected to the Internet



If a connection to another network is required, the switch may be connected through to a router, as shown in Figure 4-3.

Figure 4-3.Simple Solaris network for a single Class C network connected to another network



This enables all packets to be passed from *chardon*, *blanc*, *riesling*, and *semillon* to the switch, and through to the "internal" interface of the router. Alternatively, one of the hosts, such as *chardon*, may have a modem attached to one of its serial ports, through which an Internet connection is established. If *blanc*, *riesling,* and *semillon* wish to have direct Internet access, without telneting to *chardon*, they have to register *chardon* as their gateway. The switch would ensure that the packets were delivered to the correct gateway.

In addition, multiple hubs and switches may be daisy-chained to connect remote rooms, floors, or buildings to the same network. No more than three "hops" should exist between a router and its remotest client; otherwise, the number of packet collisions will become unacceptably high.

Most sites start with a Class C network, then begin to host multiple Class C networks, which must be connected using a router. Before we examine how to install and configure a router, let's look at the configuration of individual network interfaces more closely.

Configuring Network Interfaces

Although the various Solaris installation programs will happily configure built-in network interfaces at installation, there are several situations where you may need to add another interface or modify the configuration of the existing interfaces. These situations include:

- Setting up an existing host as a router
- Relocating a host to a different subnet
- Setting up load balancing across different interfaces

In order to enable a network interface under Solaris, several steps may be necessary. These include:

- Installing any device drivers
- Reconfiguring the system by rebooting
- Assigning an IP address to the interface

- Deciding whether the interface acts as a router component or as a component of a multi-homed host
- Creating a hosts entry that maps the IP address to a hostname
- Configuring and plumbing the interface for passing traffic

Device drivers are typically stored in */kernel/drv* (or as defined in */etc/system*) and listed in */etc/device_aliases*. For example, the standard quad ethernet connector supplied by Sun has the driver */kernel/drv/qfe*, and has its alias listed in */etc/device_aliases* as *qfe SUNW,qfe*. Rebooting with the following command forces a reconfiguration reboot:

```
bash-2.03# touch /reconfigure; init 6
```

Alternatively, from the OpenBoot PROM monitor, the following command can be used to force a reconfiguration boot:

```
OK boot -r
```

An IP address is assigned to the interface by inserting the IP address into a hostname file, located in the */etc* directory. For a system with a single interface (e.g., */dev/eri0*), such as the Blade 100, the hostname file is called *hostname.eri0*, where *eri* is the device name and 0 is the interface number.

Alternatively, a quad ethernet card (with devices */dev/qfe0*, */dev/qfe1*, */dev/qfe2*, and */dev/qfe3*) would have four hostname files containing distinct IP addresses: *hostname.qfe0*, *hostname.qfe1*, *hostname.qfe2*, and *hostname.qfe3*. These may be allocated sequentially, such as `192.64.18.1`, `192.64.18.2`, `192.64.18.3`, and `192.64.18.4`, if the host is multi-homed, or distinctly, where the system acts as a router rather than a multi-homed host.

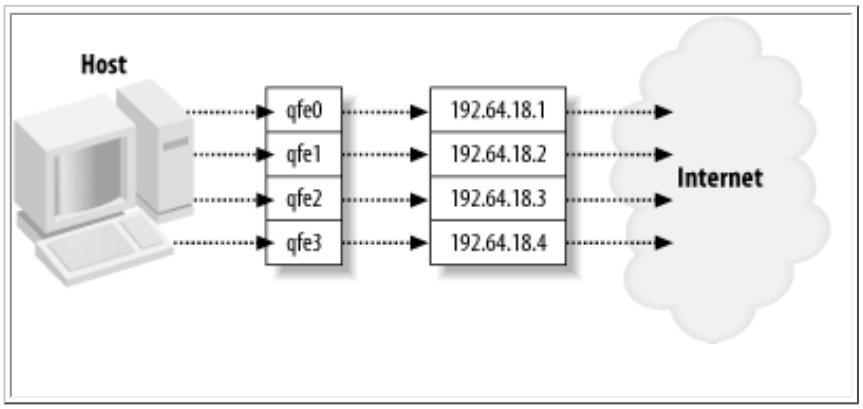
A mulit-homed host allows data to be exchanged only on the local area network (including with the router defined for that network), while a router is responsible for conveying packets between networks. To prevent routing, a multi-homed host must touch the file */etc/notrouter*. In addition, the default router for the local network should have its IP address inserted into the file */etc/defaultrouter*.

You can create a hosts entry for each interface in the */etc/hosts* file or by inserting a record into whatever distributed naming service is mandated by */etc/nsswitch.conf*. For example, if the IP address contained in *hostname.qfe0*, *hostname.qfe1*, *hostname.qfe2*, and *hostname.qfe3* were to be mapped to the hostnames *www1*, *www2*, *www3,* and *www4*, the */etc/hosts* file would contain the following entries:

```
bash-2.03# cat /etc/hosts
www1        192.64.18.1
www2        192.64.18.2
www3        192.64.18.3
www4        192.64.18.4
```

Figure 4-4 shows a logical configuration of a quad Ethernet card in a single host, operating as four independent web servers.

Figure 4-4.Logical configuration of a quad Ethernet card



Alternatively, if DNS is being used (as shown in Chapter 5), the following entries would need to be made in the appropriate zone file:

```
www1    IN    A    192.64.18.1    ;webserver
www2    IN    A    192.64.18.2    ;webserver
www3    IN    A    192.64.18.3    ;webserver
www4    IN    A    192.64.18.4    ;webserver
```

The *ifconfig* command is used to plumb and configure each interface, so that it can pass and receive IP traffic. Once the interface has been enabled, the *ifconfig* command can be used to view all active interfaces:

```
bash-2.03# /usr/sbin/ifconfig -a
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
eri0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 10.64.18.3 netmask ffffff00 broadcast 10.64.18.255
lo0: flags=2000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6> mtu 8252 index 1
        inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6> mtu 1500 index 2
        inet6 fe80::203:baff:fe04:a4e8/10
```

If an interface is configured incorrectly, the following error message will be displayed for each interface that is checked individually using *ifconfig*:

```
bash-2.03# ifconfig eri0
ifconfig: status: SIOCGLIFFLAGS: eri0: no such interface
```

Assuming that the *eri0* device is installed correctly, with the appropriate device drivers, the following *ifconfig* command should configure the device at the hardware level:

```
bash-2.03# /usr/sbin/ifconfig eri0 plumb
```

Once the device is plumbed, its runtime parameters, such as its IP address, can also be configured by using the *ifconfig* command:

```
bash-2.03# /usr/sbin/ifconfig eri0 10.64.18.3 broadcast 10.64.18.255 netmask 255.255.
    255.0
```

To bring up the interface, the *up* keyword must be used:

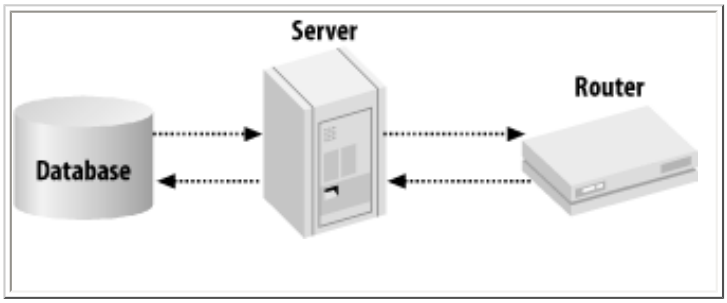
```
bash-2.03# /usr/sbin/ifconfig eri0 up
```

All of these individual commands can be combined into the following command, which configures the hardware, sets all parameters, and brings up the interface:

```
bash-2.03# /usr/sbin/ifconfig eri0 10.64.18.3 broadcast 10.64.18.255 netmask 255.255.
    255.0 plumb up
```

Depending on your local network configuration, it might be appropriate to create a point-to-point connection, rather than the previous generic connection. For example, if we want to restrict access to a secure database system, we might create a point-to-point connection that allows access to the database from only the host to which it is directly connected, as shown in Figure 4-5. In this scenario, the database connects to a server, which then connects to the router; no traffic can pass directly from the router to the database system without first passing through the intermediate server. Thus, if a hacker wanted to break into the database system, he would need to breach both the router and the server system.

Figure 4-5.Securing a database system by point-to-point networking



In order to determine whether the interfaces are being addressed correctly by other hosts on the local network, use the *arp* command to display all active connections between the localhost and other hosts:

```
bash-2.03# /usr/sbin/arp -a

Net to Media Table: IPv4
Device   IP Address               Mask      Flags   Phys Addr
------ -------------------- --------------- ----- ---------------
eri0   hp                   255.255.255.255       00:50:ba:13:08:18
eri0   austin               255.255.255.255 SP    00:03:ba:04:a4:e8
eri0   224.0.0.0            240.0.0.0       SM    01:00:5e:00:00:00
```

This displays the ethernet address to IP address mapping for the local host. The flags displayed include:

P

Published address

S

Static address

U

Unresolved address

M

Mapping address for multicast

Finally, it may be necessary to set some protocol transmission parameters manually to achieve optimal performance. Use the *ndd* command to set parameters for TCP, UDP, ARP, and IP. In addition, *ndd* can be used to display the list of all current parameter values relating to a specific protocol. For example, to display the parameters currently associated with TCP, use the following command:

```
bash-2.03# ndd /dev/tcp \?
?                               (read only)
tcp_close_wait_interval         (read and write)
tcp_conn_req_max_q              (read and write)
tcp_conn_req_max_q0             (read and write)
tcp_conn_req_min                (read and write)
tcp_conn_grace_period           (read and write)
tcp_cwnd_max                    (read and write)
tcp_debug                       (read and write)
tcp_smallest_nonpriv_port       (read and write)
tcp_ip_abort_cinterval          (read and write)
tcp_ip_abort_linterval          (read and write)
tcp_ip_abort_interval           (read and write)
tcp_ip_notify_cinterval         (read and write)
tcp_ip_notify_interval          (read and write)
tcp_ip_ttl                      (read and write)
tcp_keepalive_interval          (read and write)
tcp_maxpsz_multiplier           (read and write)
tcp_mss_def                     (read and write)
tcp_mss_max                     (read and write)
tcp_mss_min                     (read and write)
tcp_naglim_def                  (read and write)
tcp_rexmit_interval_initial     (read and write)
tcp_rexmit_interval_max         (read and write)
tcp_rexmit_interval_min         (read and write)
tcp_wroff_xtra                  (read and write)
tcp_deferred_ack_interval       (read and write)
tcp_snd_lowat_fraction          (read and write)
tcp_sth_rcv_hiwat               (read and write)
tcp_sth_rcv_lowat               (read and write)
tcp_dupack_fast_retransmit      (read and write)
tcp_ignore_path_mtu             (read and write)
tcp_rcv_push_wait               (read and write)
tcp_smallest_anon_port          (read and write)
tcp_largest_anon_port           (read and write)
tcp_xmit_hiwat                  (read and write)
tcp_xmit_lowat                  (read and write)
tcp_recv_hiwat                  (read and write)
tcp_recv_hiwat_minmss           (read and write)
tcp_fin_wait_2_flush_interval   (read and write)
tcp_co_min                      (read and write)
tcp_max_buf                     (read and write)
tcp_zero_win_probesize          (read and write)
tcp_strong_iss                  (read and write)
tcp_rtt_updates                 (read and write)
tcp_wscale_always               (read and write)
tcp_tstamp_always               (read and write)
```

```
tcp_tstamp_if_wscale          (read and write)
tcp_rexmit_interval_extra     (read and write)
tcp_deferred_acks_max         (read and write)
tcp_slow_start_after_idle     (read and write)
tcp_slow_start_initial        (read and write)
tcp_co_timer_interval         (read and write)
tcp_extra_priv_ports          (read only)
tcp_extra_priv_ports_add      (write only)
tcp_extra_priv_ports_del      (write only)
tcp_status                    (read only)
tcp_bind_hash                 (read only)
tcp_listen_hash               (read only)
tcp_conn_hash                 (read only)
tcp_queue_hash                (read only)
tcp_host_param                (read and write)
tcp_1948_phrase               (write only)
```

Obtaining Network Statistics

Once all network interfaces are configured as required, use the *netstat* command, which is responsible for gathering network statistics of various types, to verify their operational status. This data is gathered by using the interfaces on the local host.

netstat is able to gather statistics for the following types of data:

- Data grouped by protocol type
- Device statistics grouped by address type, including IPv4, IPv6, and Unix addresses
- DHCP data
- Interface data grouped by multicast
- Routing table details (including multicast)
- STREAMS data
- The state of all available IP interfaces
- The state of all active sockets, routes, physical interfaces, and logical interfaces

In the following sections, we'll review each of these data gathering operations and discuss how each is used to aid in troubleshooting and pinpointing performance issues.

Protocol Statistics

The per-protocol statistics can be divided into several categories:

RAWIP (raw IP) packets	TCP packets
IPv4 packets	ICMPv4 packets

IPv6 packets ICMPv6 packets

UDP packets IGMP packets

Each packet type has a specific set of measures associated with it. For example, RAWIP packets have counters that check the number of input (*rawipInDatagrams*) and output (*rawipOutDatagrams*) datagrams received since boot. UDP has a corrsponding set of counters that measure the number of input (*udpInDatagrams*) and output (*udpOutDatagrams*) datagrams received since boot. In addition to counters of normal events, *netstat* reports on error events, such as the number of UDP input (*udpInErrors*) and the number of UDP output (*udpOutErrors*) errors. These values should be monitored regularly to ensure that the ratio of error to normal conditions does not increase over time. For example, there are 293 *tcpActiveOpens* shown in the following listing, compared to only one *tcpAttemptFails* event. If the ratio of *tcpAttemptFails* to *tcpActiveOpens* increases over time for TCP traffic, the appropriate TCP parameters may need to be modified by using *ndd*, or a network error may need to be diagnosed. Here's a representative set of examples for understanding per-protocol errors for IPv6.

```
bash-2.03$ netstat -s

IPv6    ipv6Forwarding      =     2     ipv6DefaultHopLimit =   255
        ipv6InReceives      =     0     ipv6InHdrErrors     =     0
        ipv6InTooBigErrors  =     0     ipv6InNoRoutes      =     0
        ipv6InAddrErrors    =     0     ipv6InUnknownProtos =     0
        ipv6InTruncatedPkts =     0     ipv6InDiscards      =     0
        ipv6InDelivers      =    25     ipv6OutForwDatagrams=     0
        ipv6OutRequests     =    42     ipv6OutDiscards     =     2
        ipv6OutNoRoutes     =     0     ipv6OutFragOKs      =     0
        ipv6OutFragFails    =     0     ipv6OutFragCreates  =     0
        ipv6ReasmReqds      =     0     ipv6ReasmOKs        =     0
        ipv6ReasmFails      =     0     ipv6InMcastPkts     =     0
        ipv6OutMcastPkts    =    14     ipv6ReasmDuplicates =     0
        ipv6ReasmPartDups   =     0     ipv6ForwProhibits   =     0
        udpInCksumErrs      =     0     udpInOverflows      =     0
        rawipInOverflows    =     0     ipv6InIPv4          =     0
        ipv6OutIPv4         =     0     ipv6OutSwitchIPv4   =     0

ICMPv6  icmp6InMsgs         =     0     icmp6InErrors       =     0
        icmp6InDestUnreachs =     0     icmp6InAdminProhibs =     0
        icmp6InTimeExcds    =     0     icmp6InParmProblems =     0
        icmp6InPktTooBigs   =     0     icmp6InEchos        =     0
        icmp6InEchoReplies  =     0     icmp6InRouterSols   =     0
        icmp6InRouterAds    =     0     icmp6InNeighborSols =     0
        icmp6InNeighborAds  =     0     icmp6InRedirects    =     0
        icmp6InBadRedirects =     0     icmp6InGroupQueries =     0
        icmp6InGroupResps   =     0     icmp6InGroupReds    =     0
        icmp6InOverflows    =     0
        icmp6OutMsgs        =     8     icmp6OutErrors      =     0
        icmp6OutDestUnreachs=     0     icmp6OutAdminProhibs=     0
        icmp6OutTimeExcds   =     0     icmp6OutParmProblems=     0
        icmp6OutPktTooBigs  =     0     icmp6OutEchos       =     0
        icmp6OutEchoReplies =     0     icmp6OutRouterSols  =     3
        icmp6OutRouterAds   =     0     icmp6OutNeighborSols=     1
        icmp6OutNeighborAds =     0     icmp6OutRedirects   =     0
        icmp6OutGroupQueries=     0     icmp6OutGroupResps  =     4
```

```
        icmp6OutGroupReds    =      0
```

Address Type Statistics

The per-address statistics can be divided into three categories:

- inet (AF_INET)
- inet6 (AF_INET6)
- unix (AF_UNIX)

Let's look at sample output from the AF_UNIX sockets:

```
bash-2.03$ netstat -f unix

Active UNIX domain sockets
Address     Type       Vnode       Conn     Local Addr      Remote Addr
30000d03738 stream-ord 30000d1eb78 00000000 /tmp/.X11-unix/X0
30000d038e0 stream-ord 00000000    00000000
30000d03a88 stream-ord 30000ce4a30 00000000 /tmp/jd_sockV6
30000d03c30 stream-ord 30000a62d78 00000000 /dev/kkcv
30000d03dd8 stream-ord 30000a62f50 00000000 /dev/ccv
```

Here we can see a number of different active sockets using Unix type addressing, such as the X11 server, which has the address `30000d03738`.

Multicast Statistics

The multicast statistics option provides an overview of interfaces that are currently listening for multicast broadcasts on the `224.0.0.1` (ALL_HOSTS) address. This is so that packets can be routed appropriately using the router discovery daemon (*in.rdisc*), discussed in the next section, "Routing." In the following example, both the IPv4 and IPv6 multicast groups are displayed:

```
bash-2.03$ netstat -g

Group Memberships: IPv4
Interface Group               RefCnt
--------- ------------------- ------
lo0       224.0.0.1                1
eri0      224.0.0.1                1

Group Memberships: IPv6
If    Group                   RefCnt
----- ----------------------- ------
lo0   ff02::1:ff00:1                1
lo0   ff02::1                      1
eri0  ff02::202                    1
eri0  ff02::1:ff04:a4e8            1
eri0  ff02::1                      2
```

Routing Statistics

The kernel maintains a table of routes, constructed by the routing daemon, *in.routed*. The various routes that have been configured are always viewable by checking the routing statistics:

```
bash-2.03$ netstat -r

Routing Table: IPv4
Destination           Gateway              Flags Ref   Use    Interface
-------------------- -------------------- ----- ----- ------ ---------
10.64.18.0           austin                U         1      5  eri0
224.0.0.0            austin                U         1      0  eri0
localhost            localhost             UH       25 215051  lo0
```

Here, we can see there are two network routes available for packets on the primary Ethernet interface *eri0*: the `10.64.18.0` network and the `224.0.0.0` multicast network. In addition, the loopback interface (*lo0*) has the local host interface, which is commonly used for troubleshooting and testing. These routes are all IPv4; however, IPv6 routing details are also displayed:

```
Routing Table: IPv6
Destination/Mask            Gateway                     Flags Ref Use    If
--------------------------- --------------------------- ----- --- ------ -----
fe80::/10                   fe80::203:baff:fe04:a4e8    U       1      0 eri0
ff00::/8                    fe80::203:baff:fe04:a4e8    U       1      0 eri0
default                     fe80::203:baff:fe04:a4e8    U       1      0 eri0
localhost                   localhost                   UH      5     28 lo0
```

STREAMS Statistics

STREAMS is a System V package that provides access to system calls, standard libraries, and the kernel for the purposes of writing network applications. Any application that uses STREAMS has a specific set of properties about which statistics can be collected, since the I/O operations are distinct from other networking APIs (such as the BSD-style socket API). *netstat* reports these statistics, including queues, which comprise the read/write operations that characterize a stream:

```
bash-2.03$ netstat -m
streams allocation:
                                      cumulative  allocation
                     current   maximum       total    failures
streams                  326       340        7634           0
queues                   938       962       18662           0
mblk                    1144      1651        7773           0
dblk                    1140      1729     2349590           0
linkblk                   11       169          18           0
strevent                   9       169      121739           0
syncq                     25        50         101           0
qband                      0         0           0           0

1646 Kbytes allocated for streams data
```

More details can be obtained by reading the manpage for *streamio*.

IP Interface Statistics

netstat also reports statistics obtained at the IP level. This includes the number of input and output packets

counted, the number of input and output errors, and the number of packet collisions. Again, separate entries are shown for IPv4 and IPv6:

```
bash-2.03$ netstat -i
Name  Mtu  Net/Dest      Address        Ipkts  Ierrs Opkts  Oerrs Collis Queue
lo0   8232 loopback      localhost      227695 0     227695 0     0      0
eri0  1500 austin        austin         2573   0     2130   0     0      0

Name  Mtu  Net/Dest      Address         Ipkts  Ierrs Opkts  Oerrs Collis
lo0   8252 localhost     localhost       227705 0     227705 0     0
eri0  1500 fe80::203:baff:fe04:a4e8/10 fe80::203:baff:fe04:a4e8    2573   0
2130   0     0
```

Combined Socket, Route, and Interface Statistics

Most administrators prefer to combine the information that *netstat* provides into a single report-style format. This can be achieved by using the combined route, socket, and interface statistics, as shown in the output in Example 4-1.

Example 4-1: Output of the netstat-a command

```
bash-2.03$ netstat -a
UDP: IPv4
   Local Address         Remote Address     State
-------------------- -------------------- -------
      *.route                              Idle
      *.*                                  Unbound
      *.*                                  Unbound
      *.sunrpc                             Idle
      *.*                                  Unbound
      *.32771                              Idle
      *.sunrpc                             Idle
      *.*                                  Unbound
      *.32775                              Idle
      *.32779                              Idle
      *.32780                              Idle
Routing
*.*                                  Unbound
      *.32821                              Idle
      *.32822                              Idle
      *.32823                              Idle
      *.name                               Idle
      *.biff                               Idle
      *.talk                               Idle
      *.time                               Idle
      *.echo                               Idle

UDP: IPv6
   Local Address                 Remote Address                   State
If
--------------------------------- ---------------------------------- ---------- -
----
      *.*                                                          Unbound
      *.sunrpc                                                     Idle
```

```
      *.*                                                    Unbound
      *.32771                                                Idle
      *.32779                                                Idle
      *.*                                                    Unbound
      *.32821                                                Idle
      *.time                                                 Idle

TCP: IPv4
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q  State
-------------------- -------------------- ----- ------ ----- ------ -------
      *.*                  *.*                0      0 24576      0 IDLE
      *.sunrpc             *.*                0      0 24576      0 LISTEN
      *.*                  *.*                0      0 24576      0 IDLE
      *.sunrpc             *.*                0      0 24576      0 LISTEN
      *.*                  *.*                0      0 24576      0 IDLE
      *.32775              *.*                0      0 24576      0 LISTEN
      *.32776              *.*                0      0 24576      0 LISTEN
      *.32782              *.*                0      0 24576      0 LISTEN
      *.32783              *.*                0      0 24576      0 LISTEN

TCP: IPv6
Local Address   Remote Address  Swind  Send-Q  Rwind  Recv-Q  State  If
*.*             *.*             0              24576  0       IDLE
*.sunrpc        *.*             0      0       24576  0       LISTEN
*.*             *.*             0      0       24576  0       IDLE
*.32775         *.*             0      0       24576  0       LISTEN
localhost.32780 localhost.32775 32768  0       32768  0       CLOSE_WAIT
*.32782         *.*             0      0       24576  0       LISTEN
*.32791         *.*             0      0       24576  0       LISTEN
*.ftp           *.*             0      0       24576  0       LISTEN
*.telnet        *.*             0      0       24576  0       LISTEN

Active UNIX domain sockets
Address  Type          Vnode     Conn  Local Addr      Remote Addr
30000d03738 stream-ord 30000d1eb78 00000000 /tmp/.X11-unix/X0
30000d038e0 stream-ord 00000000 00000000
30000d03a88 stream-ord 30000ce4a30 00000000 /tmp/jd_sockV6
30000d03c30 stream-ord 30000a62d78 00000000 /dev/kkcv
30000d03dd8 stream-ord 30000a62f50 00000000 /dev/ccv
```

Some of the TCP messages shown in this output, for both IPv4 and IPv6, may be unfamiliar, so we review each of them individually in Table 4-1.

Table 4-1:TCP constants reported by netstat

Message	Description
BOUND	Socket is bound.
CLOSED	Socket is closed.
CLOSING	Socket is closing.

CLOSE_WAIT	Socket is waiting to close.
ESTABLISHED	Socket has connected successfully.
FIN_WAIT_1	Socket is closing (local).
FIN_WAIT_2	Socket is closing (remote).
IDLE	Socket is idle.
LAST_ACK	Socket will close after receiving last acknowledgment.
LISTEN	Socket is active and listening.
SYN_RECEIVED	Socket is being synchronized.
SYN_SENT	Socket is creating a connection.
TIME_WAIT	Socket is waiting to close.

Routing

Imagine that you are a courier, and your run always starts at the local courier depot. You're given a list of addresses, which are associated with a set of packages, and your goal is to deliver them in as little time as possible, subject to the following constraints:

- The number of roads you take to deliver each package must be minimized.
- You must avoid deadends and accidents.
- You can only determine which roads to take by consulting a street directory and by crosschecking street names along your path with those in your directory.

If this seems like a fairly trivial task for a courier, consider how much more difficult the job would be if the following conditions prevailed:

- The number of possible roads increases exponentially each year. You might be asked to take roads you've never heard of before!
- There is no way of knowing, in advance, where accidents or deadends might occur.
- The street directory you have is completely out of date, because the number of highways increases exponentially each year.

This scenario describes the difficulties faced by the emergence of the Internet and the massive interconnections between hosts and networks. In order for a packet of data to be transferred from host A to host B, a physical path must be identified for the packet to travel.

There is no central lookup service that decides how to route each packet between all possible combinations of two hosts on the Internet (i.e., between the sender and the receiver). This means routes must be generated dynamically. (The only exceptions to this rule are certain situations where a predictable static route may be installed.)

When transferring data around the Internet or between subnets, intermediate hosts must be responsible for transferring packets between networks; these hosts are called routers and are responsible for routing packets between hosts, which can be separated by single subnets or by entire continents. To gain insight into how many routers a packet transfer may involve, let's use the *traceroute* command to display the "hops" required to connect from a host in Sydney, Australia, to the Sun Microsystems web server:

```
bash-2.03$ traceroute wwwwseast.usec.sun.com/
Tracing route to wwwseast.usec.sun.com [192.9.49.30]
over a maximum of 30 hops:
  1   184 ms   142 ms   138 ms  202.10.4.131
  2   147 ms   144 ms   138 ms  202.10.4.129
  3   150 ms   142 ms   144 ms  202.10.1.73
  4   150 ms   144 ms   141 ms  atm11-0-0-11.ia4.optus.net.au [202.139.32.17]
  5   148 ms   143 ms   139 ms  202.139.1.197
  6   490 ms   489 ms   474 ms  hssi9-0-0.sf1.optus.net.au [192.65.89.246]
  7   526 ms   480 ms   485 ms  g-sfd-br-02-f12-0.gn.cwix.net [207.124.109.57]
  8   494 ms   482 ms   485 ms  core7-hssi6-0-0.SanFrancisco.cw.net [204.70.10.9]
  9   483 ms   489 ms   484 ms  corerouter2.SanFrancisco.cw.net [204.70.9.132]
 10   557 ms   552 ms   561 ms  xcore3.Boston.cw.net [204.70.150.81]
 11   566 ms   572 ms   554 ms  sun-micro-system.Boston.cw.net [204.70.179.102]
 12   577 ms   574 ms   558 ms  wwwwseast.usec.sun.com [192.9.49.30]
Trace complete.
```

Here, we can see that some 12 hosts are required to transfer packets between the sender and the receiver. In addition, the observed response times can be quite slow--often more than half a second. It is possible for attempted connections to time out. This can be very useful when trying to identify which intermediate host and/or network is having problems when your remote connection to a host half a world away suddenly dies!

In this section, we'll examine how Solaris solves a number of the classic routing problems.

Static routing typically involves creating a direct physical connection between two hosts, where the implementation of dynamic routing would be wasteful or a security risk. For example, if your local network has three subnets that need to share data, a static route could be created between each router and the other two routers in the network. The number of specific routes required to allow data to flow seamlessly between networks is directly proportional to the square of the number of routers on the network. Every time a change is made to the network topology, these routes will have to be modified manually. If that sounds like too much hard work, consider the situation where it might be desirable: a secure database server that can be accessible only by knowing the route to the host and is not publicly announced. Instead of permitting route discovery, a static route is an appropriate technique here. This could be implemented by creating a point-to-point configuration using *ifconfig* on a secondary interface, as discussed in the network interface configuration section.

The alternative to static routing is dynamic routing, which involves two daemons: the routing daemon proper (*in.routed*) and the route discovery daemon (*in.rdisc*). The *in.routed* daemon implements the Routing Information Protocol, and is responsible for updating and managing entries in the kernel's routing tables. It uses UDP (port 520) for performing routing operations and operates on all network interfaces that have been plumbed and are identified as up.

If the */etc/notrouter* file does not exist, and given that two or more operational interfaces can be found, the

host begins to act as a router. Data can then be exchanged between data received on one interface, destined to be transmitted from another interface. For a local area network, the interface that connects to all local hosts is usually known as the *internal* interface, while the interface that is visible downstream to an ISP or another subnet is known as the *external* interface. By using packet filtering, it is possible to specify a set of rules governing what type (TCP or UDP) of packets can be transferred between interfaces and on which ports. This is obviously important for protecting local networks, since services that are available to local hosts may not be appropriate for public access.

The route discovery daemon, *in.rdisc*, implements the Internet Control Message Protocol (ICMP). In terms of route discovery, *in.rdisc* running on host systems listens for multicast broadcasts on the `224.0.0.1` (ALL_HOSTS) address. These messages are prioritized, and the default router is selected based on its proximity to the host. On routers, *in.rdisc* broadcasts its availability using multicast on `224.0.0.1`, and listens for requests on `224.0.0.2` (ALL_ROUTERS). Hosts may request a router directly by broadcasting on `224.0.0.2`.

Back to: Solaris 8 Administrator's Guide
