Chapter 18
Kernel Interface

18.1 Kernel Interface Overview

Although the kernel interface is not technically a routing protocol, it has many characteristics of one, and GateD handles it similarly. The routes GateD chooses to install in the kernel forwarding table are those that will actually be used by the kernel to forward packets.

The add, delete, and change operations that GateD must use to update the typical kernel forwarding table take a non-trivial amount of time. The time used does not present a problem for older routing protocols (such as RIP), which are not particularly time critical and do not easily handle large numbers of routes anyway. The newer routing protocols (such as OSPF and BGP) have stricter timing requirements and are often used to process many more routes. The speed of the kernel interface becomes critical when these protocols are used.

To prevent GateD from locking up for significant periods of time while installing large numbers of routes (up to a minute or more has been observed on real networks), the processing of these routes is done in batches. The size of these batches can be controlled by the tuning parameters shown below, but normally the default parameters will provide the proper functionality.

During normal shutdown processing, GateD deletes all the routes it has installed in the kernel forwarding table, except for those static routes marked with **`retain`**. Optionally, GateD can leave all routes in the kernel forwarding table by not deleting any routes using **`noflushatexit`**. This option is useful on systems with large numbers of routes because it eliminates the need to re-install the routes when GateD restarts, which can greatly reduce the time it takes to recover from a restart.

18.2 Kernel Interface Syntax

```
kernel {
    [ options
        [ nochange ]
        [ noflushatexit ]
    ; ]
    [ remnantholdtime time ; ]
    [ routes number ; ]
    [ flash
        [ limit number ]
        [ type ( interface | interior | all ) ]
```

```
        ; ]
        [ background
            [ limit number ]
            [ priority ( flash | higher | lower ) ]
        ; ]
    [ traceoptions
            [ tracefile [ replace ]
            [ size tracesize [ k | m ] files tracefiles] ] [ nostamp ]
            [ trace_global_options | trace_protocol_options |
              trace_protocol_packets ]
            [ except ( trace_global_options | trace_protocol_options |
              trace_protocol_packets ) ]
        ; ]
} ;
```

More detailed descriptions of these commands can be found on page 321 of the *Command Reference Guide*.

18.3 Forwarding Tables and Routing Tables

The rest of this section assumes that the reader understands how GateD interacts with a UNIX system.

The forwarding table, also known as the forwarding information base (FIB), is the table that controls the forwarding of packets in the kernel. The routing table, also known as the routing information base (RIB), is the table that GateD uses internally to store routing information that it learns from routing protocols. The routing table is used to collect and store routes from various protocols. For each unique combination of network and mask, an active route is chosen. This route will be the one with the best (numerically smallest) preference. All the active routes are installed in the kernel forwarding table. The entries in this table are what the kernel actually uses to forward packets.

18.3.1 Updating the Forwarding Table

Two main methods of updating the kernel FIB are the **`ioctl()`** interface and the routing socket interface.

18.3.1.1 The `ioctl()` Interface

The **`ioctl()`** interface to the forwarding table was introduced in *BSD 4.3* and widely distributed in *BSD 4.3*. It has several limitations, including:

- fixed subnet masks
- a one-way interface
- blind updates
- the inability to support changes

Fixed Subnet Masks

The **`ioctl()`** interface allows only fixed subnet masks. The *BSD 4.3* networking code assumed that all subnets of a given network had the same subnet mask. This limitation is

enforced by the kernel. The network mask is not stored in the kernel forwarding table, but determined when a packet is forwarded by searching for interfaces on the same network.

One-way Interface

Because of the one-way interface, GateD is able to update the kernel forwarding table, but it is not aware of other modifications of the forwarding table. GateD is able to listen to ICMP messages and guess how the kernel has updated the forwarding table in response to ICMP redirects.

Blind Updates

Because of blind updates, GateD is not able to detect changes to the forwarding table resulting from the use of the route command by the system administrator. Use of the route command on systems that use the **`ioctl()`**interface is strongly discouraged while GateD is running.

No Change

Because no change operation is supported, a route must be deleted and a new one added to change a route that exists in the kernel.

18.3.1.2 The Routing Socket Interface

The routing socket interface to the kernel forwarding table was introduced in *BSD 4.3 Reno*, widely distributed in *BSD 4.3 Net/2,* and improved in *BSD 4.4*. This interface is simply a socket, similar to a UDP socket, on which the kernel and GateD exchange messages. It has several advantages over the **`ioctl()`** interface, including:

- variable subnet masks
- a two-way interface
- visible updates
- the ability to support changes
- the ability to be expanded

Variable Subnet Masks

Variable subnet masks are different masks that can be used on the subnets of the same network. Because the network mask is passed to the kernel explicitly, these variable subnet masks can be used. Also, routes with masks that are more general than the natural mask can be used. Using more general masks is known as “classless” routing.

Two-way Interface

A two-way interface allows GateD to change the kernel forwarding table with this interface and allows the kernel to report changes to the forwarding table to GateD. A redirect message that has modified the kernel forwarding table can now be reported, which means that GateD no longer needs to monitor ICMP messages to learn about redirect messages. Also, the kernel now indicates whether it processed the redirect message, which allows GateD to safely ignore redirect messages that the kernel did not process.

Visible Updates

Visible updates allow changes to the routing table by other processes, including the route command, to be received via the routing socket. Because these changes are received, GateD can ensure that the kernel forwarding table is in sync with the routing table. Also, the system administrator can use the **route** command while GateD is running.

Changes

The ability to support changes allows routes in the kernel to be atomically changed. (Because some early versions of the kernel routing socket code had bugs in the change message processing, there are compilation time and configuration time options that cause delete and add sequences to be used in lieu of change messages.)

Expansion

The ability to be expanded allows new levels of kernel/GateD communications to be added by adding new message types.

18.3.2 Reading the Forwarding Table

When GateD starts up, it reads the kernel forwarding table and installs corresponding routes into the routing table. These routes are called “remnants” and are timed out after a three-minute interval, or as soon as a more attractive route is learned. This system allows forwarding to occur while the routing protocols start learning routes.

Three main methods for reading the forwarding table from the kernel are via:

- kmem
- getkerninfo/sysctl
- OS-specific methods

18.3.2.1 Reading Forwarding Table via kmem

On many systems, especially those based on *BSD 4.3*, GateD must have knowledge of the kernel's data structures to read the current state of the forwarding table. This method is slow and subject to error if the kernel forwarding table is updated while GateD is in the middle of reading it. Errors are likely to occur if the system administrator uses the **route** command, or if an ICMP redirect message is received while GateD is starting up.

Due to an oversight, some systems, such as *OSF/1*, which are based on *BSD 4.3 Reno* or later, do not have the **getkerninfo()** system call described below, which allows GateD to read routes from the kernel without knowing about kernel internal structures. On these systems, it is necessary to read the kernel radix tree from the kernel by reading kernel memory. Reading the radix tree is even more error prone than reading the hash-based forwarding table.

18.3.2.2 Reading the Forwarding Table via getkerninfo/sysctl

Besides the routing socket, *BSD 4.3 Reno* introduced the **getkerninfo()** system call. This call allows a user process (such as GateD) to read various information from the kernel without knowledge of the kernel data structures. In the case of the forwarding table, it is returned to GateD automatically as a series of routing socket messages. This method pre-

vents the problems associated with the forwarding table changing while GateD is reading it.

BSD 4.4 changed the **getkerninfo()** interface into the **sysctl()** interface, which takes different parameters, but otherwise functions identically.

18.3.2.3 Reading the Forwarding Table via OS-specific Methods

Some operating systems, for example *SunOS 5*, define their own method of reading the kernel forwarding table. The *SunOS 5* version is similar in concept to the **getkerninfo()** method.

18.4 Reading the Interface List

The kernel support subsystem of GateD is responsible for reading the status of the kernel's physical and protocol interfaces periodically. GateD detects changes in the interface list and notifies the protocols so that they can start or stop instances or peers. The interface list is read one of the following two ways:

- SIOCGIFCONF
- sysctl

18.4.1 Reading the Interface List with SIOCGIFCONF

On systems based on *BSD 4.3*, *4.3 Reno* and *4.3 Net/2*, the **SIOCGIFCONF ioctl** interface is used to read the kernel interface list. Using this method, a list of interfaces and some basic information about them is returned by the SIOCGIFCONF call. Other information must be learned by issuing other ioctls to learn the interface network mask, flags, MTU, metric, destination address (for point-to-point interfaces), and broadcast address (for broadcast capable interfaces).

GateD reads and re-reads this list every 15 seconds, looking for changes. When the routing socket is in use, GateD also re-reads the list whenever a message is received, indicating a change in routing configuration. Receipt of a **SIGUSR2** signal also causes GateD to re-read the list. The interval in which GateD reads the list can be explicitly configured in the interface configuration. (See “Chapter 7 Interface Statement” on page 23 for more information about the interface statement.)

18.4.2 Reading the Interface List with sysctl

BSD 4.4 added the ability to read the kernel interface list via the sysctl system call. The interface status is returned automatically as a list of routing socket messages that GateD parses for the required information.

BSD 4.4 also added routing socket messages to report interface status changes immediately. This allows GateD to react quickly to changes in interface configuration.

When **sysctl** is used, GateD re-reads the interface list only once a minute. It also re-reads it on routing table change indications and when a **SIGUSR2** is received. This interval can be explicitly configured in the interface configuration. (See “Chapter 7 Interface Statement” on page 23 for more information about the interface statement.)

18.5 Reading Interface Physical Addresses

Later versions of the **getkerninfo()** and **sysctl()** interfaces return the interface physical addresses as part of the interface information. On most systems where information about physical addresses is not returned, GateD scans the kernel physical interface list for this information for interfaces with **IFF_BROADCAST** set, assuming that their drivers are handled the same as Ethernet drivers. On some systems, such as *SunOS 4* and *SunOS 5*, system-specific interfaces are used to learn this information.

The interface physical addresses are useful for IS-IS. For IP protocols, they are not currently used, but may be in the future.

18.6 Reading Kernel Variables

At startup, GateD reads some special variables out of the kernel, which is usually done with the **nlist** (or **kvm_nlist**) system call. Some systems use different methods.

The variables read include the status of UDP checksum creation and generation, IP forwarding, and kernel version (for informational purposes). On systems where the routing table is read directly from kernel memory, the root of the hash table or radix tree routing table is read. On systems where interface physical addresses are not supplied by other means, the root of the interface list is read.

18.7 Special Route Flags

The later *BSD*-based kernel supports the special route flags described below:

RTF_REJECT

Instead of forwarding a packet as with a normal route, routes with **RTF_REJECT** cause packets to be dropped and **unreachable** messages to be sent to the packet originators. This flag is valid only on routes pointing at the loopback interface.

RTF_BLACKHOLE

Like the **RTF_REJECT** flag, routes with **RTF_BLACKHOLE** cause packets to be dropped, but unreachable messages are not sent. This flag is valid only on routes pointing at the loopback interface.

RTF_STATIC

When GateD starts, it reads all the routes currently in the kernel forwarding table. Besides interface routes, it usually marks everything else as a remnant from a previous run of GateD and deletes it after a few minutes. This means that routes added with the route command will not be retained after GateD has started. To fix this, the **RTF_STATIC** flag was added. When the **route** command is used to install a route that is not an interface route, it sets the **RTF_STATIC** flag. This signals to GateD that the route was added by the system administrator and should be retained.