# Operating GateD

## V.9.3.2

nexthop

## Copyright Notice

## Trademark Notice

# Contents

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

# Chapter 1
# Starting GateD

## 1.1 Starting GateD at Boot Time

GateD is usually started from **`/etc/rc.local`** at system startup (boot) time. GateD is preferably started after the partition in which it resides is mounted, and after **`syslogd`** has been started. GateD must be started before starting any programs that require routing information; for example, NFS mounts from systems on different networks or **`xntpd`**. It does not matter whether GateD is started before or after the interfaces have been enabled with **`ifconfig`**.

## 1.2 Starting GateD Directly

Add a few lines similar to the following from **`/etc/rc.local`** to start GateD:

```
if [ -f /etc/gated.conf -a -f /usr/local/sbin/gated ]; then
     /usr/local/sbin/gated ; echo -n "gated "
fi
```

## 1.3 Starting GateD with gdc

Starting GateD with **gdc** is required when permissions are inadequate to run GateD. Again, add a few lines similar to the following from **`/etc/rc.local`** to start GateD:

```
if [ -f /etc/gated.conf -a -f /usr/local/bin/gdc ]; then
    /usr/local/bin/gdc [ flags ] start ; echo -n "gated(via gdc) " ;
fi
```

where *`flags`* are any **gdc** options. See “Chapter 3 gdc” on page 7 for more information.

## 1.4 Controlling GateD

GateD is normally controlled via the **gdc** utility, but can be controlled directly. See “Chapter 2 gated” on page 3 and “Chapter 3 gdc” on page 7 of this manual for more information.

## 1.5 Checking GateD Status

By using the **gdc** utility, you can obtain a status dump from GateD. You can query GateD for information about the RIP and OSPF protocols with the ripquery and ospfmon utilities.

# Chapter 2
# gated

## 2.1 Name

**`gated`** - gateway routing daemon

## 2.2 Synopsis

```
gated [ -c ] [ -C ] [ -n ] [ -N ] [ -t trace_options ] [ -f config_file ]
    [ trace_file ]
```

```
gated -v
```

## 2.3 Description

**`gated`** is a routing daemon that handles multiple routing protocols and replaces **`routed`** and **`egpup`**. **`gated`** currently implements many IPv4 and IPv6 routing protocols as well as some host protocols. See your NextHop customer web page or release notes for a complete listing.

## 2.4 The Command-line Options

The command-line options are:

**`-c`**

**`-c`** (lowercase) specifies that the configuration file will be parsed for syntax errors and then **`gated`** will exit. If there are no errors, **`gated`** will leave a dump file in **`/var/tmp/gated_dump`** or **`/usr/tmp/gated_dump`**. **`gated`** does not need to be run as the super user to use the **`-c`** option. If **`gated`** is not run as the super user, however, it may not be possible to read the kernel forwarding table and interface configuration. The **`-c`** option implies **`-tgeneral`**. All traceoption clauses in the configuration file will be ignored.

**`-C`**

**`-C`** (uppercase) specifies that the configuration file will be parsed only for syntax errors. **`gated`** will exit with a status **`1`** if there were errors, and **`0`** if there were not. **`gated`** does not need to be run as the super user to use the **`-C`** option. If **`gated`** is not run as the super user, however, it may not be possible to read the kernel forwarding table and interface configuration.

**`-n`**

**`-n`** specifies that **`gated`** will not modify the kernel forwarding table. **`-n`** is used for testing **`gated`** configurations with actual routing data.

**-N**

**-N** specifies that **gated** will not daemonize. Normally, if tracing to **stderr** is not specified and the parent process ID is not 1, **gated** will daemonize. This allows the use of an **/etc/inittab**-like method of invoking **gated** that does not have a Process ID of 1.

**-t** *trace_options*

**-t** *trace_options* specifies a comma-separated list of trace options to be enabled on startup. If no flags are specified, **general** is assumed. No space is allowed between this option and its arguments.

*trace_options* must be used to trace events that take place before the config file is parsed, such as determining the interface configuration and reading routes from the kernel.

See “Chapter 4 Trace Statements” on page 15 of the configuration guide for valid trace options and a more detailed explanation of tracing.

**-f** *config_file*

**-f** *config_file* specifies that **gated** will use an alternate config file. By default, **gated** uses **/etc/gated.conf**.

**-v**

**-v** specifies that **gated** will show its version information and quit.

If a trace file is specified on the command line, or no trace flags are specified on the command line, **gated** detaches from the terminal and runs in the background. If trace flags are specified without specifying a trace file, **gated** assumes that tracing is desired to **stderr** and remains in the foreground.

## 2.5 Signal Processing

The following signals may be used to control **gated**:

**SIGHUP** - reread the configuration file

A **SIGHUP** causes **gated** to reread the configuration file. **gated** first performs a clean-up of all allocated policy structures. Depending on the protocol, this may cause cycling of adjacencies and/or peering connections. GateD attempts to preserve neighbor and/or peering relationships if at all possible.

**SIGINT** - snapshot of current state

A **SIGINT** causes the current state of all **gated** tasks, timers, protocols, and tables to be written to the dump location. This may be one of two places: **/var/tmp/gated_dump** or **/usr/tmp/gated_dump**.

On systems supporting **fork()** this is done by forking a subprocess to dump the table information so as not to impact **gated**’s routing functions. On systems where memory management does not support copy-on-write, **fork()** will cause the **gated** address space to be duplicated; this can cause a noticeable impact on the system. On systems that do not support **fork(),** the main process immediately processes the dump, which can affect **gated**'s routing functions.

**SIGTERM** - graceful shutdown

On receipt of a **SIGTERM**, **gated** attempts a graceful shutdown. All tasks and protocols are asked to shutdown. Most will terminate immediately. The exception is EGP peers, which wait for confirmation. It may be necessary to repeat the **`SIGTERM`** once or twice if this process takes too long.

All protocol routes are removed from the kernel forwarding table on receipt of a **`SIGTERM`**. Interface routes, routes with **`RTF_STATIC`** set (from the route command where supported), and static routes specifying **`retain`** will remain. Use **`SIGKILL`** to terminate **`gated`** with the exterior routes intact.

**`SIGUSR1`** - toggle tracing

On receipt of a **`SIGUSR1`**, **`gated`** will close the trace file. A subsequent **`SIGUSR1`** will cause it to be reopened. This allows the file to be moved regularly. It is not possible to use **`SIGUSR1`** if a trace file has not been specified or if tracing is being performed to **`stderr`**.

**`SIGUSR2`** - check for interface changes

On receipt of a **`SIGUSR2`**, **`gated`** will rescan the kernel interface list looking for changes.

## 2.6 Files

Many of the default filenames listed below contain the string %s, which is replaced by the name with which **`gated`** is invoked. Normally this is **`gated`**, but if invoked as **`gated-test`**, **`gated`** will by default look for **`/etc/gated-test.conf`**. These paths may all be changed at compilation time.

**`/usr/tmp/gated_dump`**

**`/usr/tmp/gated_dump`** is where **`gated`** writes status information. The default is **`/usr/tmp/%s_dump`**. Another common path is **`/var/tmp/%s_dump.`**

**`/etc/gated.conf`**

**`/etc/gated.conf`** is where **`gated`** looks for its configuration file. The default is **`/etc/%s.conf.`**

**`/etc/gated.pid`**

**`/etc/gated.pid`** is where **`gated`** writes its process id (PID). The default is **`/etc/%s.pid`**, but **`/var/run/%s.pid`** is common.

# Chapter 3
# gdc

## 3.1 Name

**gdc** - operational user interface for GateD

## 3.2 Synopsis

```
gdc [ -q ] [ -n ] [ -p path ] [ -c coresize ] [ -f filesize ] [ -m datasize ]
    [ -s stacksize ] [ -t seconds ] command
```

## 3.3 Description

**gdc** provides a command-line interface for the operation of the GateD routing daemon. It provides support for:

- starting and stopping the daemon
- delivering signals to manipulate the daemon when it is operating
- maintaining and checking syntax of configuration files
- producing and removing dump and core files

**gdc** can reliably determine GateD's running state and produces a reliable exit status when errors occur, which is useful in shell scripts that manipulate GateD. Commands executed using **gdc** and, optionally, error messages produced by the execution of those commands, are logged via the same **syslogd** facility that GateD itself uses, providing an audit trail of operations performed on the daemon.

If installed as a **setuid root** program, **gdc** will allow non-root users who are members of a trusted group (by default the **gdmaint** group) to manipulate the routing daemon and deny access to others. For audit purposes, the name of the user is logged via **syslogd**, along with an indication of each command executed.

## 3.4 Command-line Options

The command-line options are:

**-n**

**-n** (lowercase) specifies to run without changing the kernel forwarding table. **-n** is useful for testing and when operating GateD as a route server that does no forwarding.

**-p**

**-p** (lowercase) specifies the path to the GateD binary. The default is the **SBINDIR** constraint (determined at compile time), appended with **gated**. Typical values are **/usr/sbin/gated** and **/sbin/gated**.

**-q**

**-q** specifies to run quietly. With this option, informational messages that are normally printed to the standard output are suppressed, and error messages are logged via **syslogd** instead of being printed to the standard error output. This is often convenient when running **gdc** from a shell script.

**-t** *seconds*

**-t** *seconds* specifies the time in seconds that **gdc** will spend waiting for GateD to complete certain operations, in particular at termination and startup. By default, *seconds* is set to 10 seconds.

These additional command-line options may be present, depending on the options used to compile **gdc**:

**-c** *coresize*

**-c** *coresize* sets the maximum size of a core dump that a GateD started with **gdc** will produce. **-c** *coresize* is useful on systems where the default maximum core dump size is too small for GateD to produce a full core dump on errors.

**-f** *filesize*

**-f** *filesize* sets the maximum file size that a GateD started with **gdc** will produce. **-f** *filesize* is useful on systems where the default maximum file dump size is too small for GateD to produce a full state dump when requested.

**-m** *datasize*

**-m** *datasize* sets the maximum size of the data segment of a GateD started with **gdc. -m** *datasize* is useful on systems where the default data segment size is too small for GateD to run.

**-s** *stacksize*

**-s** *stacksize* sets the maximum size of stack of a GateD started with **gdc**. **-s** *stacksize* is useful on systems where the default maximum stack size is too small for GateD to run.

The following commands cause signals to be delivered to GateD for various purposes:

**COREDUMP**

**COREDUMP** sends an abort signal to GateD, causing it to terminate with a core dump.

**dump**

**dump** signals GateD to dump its current state into the dump location, which is either **/var/tmp/gated_dump** or **/usr/tmp/gated_dump**.

**interface**

**interface** signals **gated** to recheck the interface configuration. GateD normally does this periodically, but **interface** can be used to force the daemon to check interface status immediately when changes are known to have occurred. This is useful when GateD is running on an operating system that does not support asynchronous notification of interface state changes (for example, via a routing socket). When an interface change is made by the administrator, this option may then be used to notify GateD of the new state immediately instead of waiting for the next poll.

**KILL**

**KILL** causes GateD to terminate ungracefully. It is more desirable to allow GateD to terminate itself gracefully by using the **term** option.

**reconfig**

**reconfig** signals GateD to reread its configuration file, reconfiguring its current state as appropriate.

**term**

**term** signals GateD to terminate after shutting down all operating routing protocols gracefully. Executing this command a second time should cause GateD to terminate even if some protocols have not yet fully shut down.

**toggletrace**

If GateD is currently tracing to a file, **toggletrace** causes tracing to be suspended and the trace file to be closed. If GateD tracing is currently suspended, **toggletrace** causes the trace file to be reopened and tracing initiated. **toggletrace** is useful for moving trace files.

By default, GateD obtains its configuration from a file normally named **/etc/gated.conf**. The **gdc** program also maintains several other versions of the configuration file, in particular those named:

**/etc/gated.conf+**

**/etc/gated.conf+** is the new configuration file. When **gdc** is requested to install a new configuration file, this file is renamed **/etc/gated.conf**.

**/etc/gated.conf-**

**/etc/gated.conf-** is the old configuration file. When **gdc** is requested to install a new configuration file, the previous **/etc/gated.conf** is renamed to this name.

**/etc/gated.conf--**

**/etc/gated.conf--** is the very old configuration file. **gdc** retains the previous old configuration file under this name.

The following commands perform operations related to configuration files:

**checkconf**

**checkconf** checks **/etc/gated.conf** for syntax errors. To ensure that there are no errors in the configuration that would cause running GateD to terminate on reconfiguration, **checkconf** is usually done after changing the configuration file but before sending a **reconfig** signal to the currently running GateD. When **checkconf** is used, **gdc** issues an informational message indicating whether there were parse errors, and, if so, saves the error output in a file for inspection.

**checknew**

Like **checkconf** except that the new configuration file, **/etc/gated.conf+**, is checked instead.

**newconf**

**newconf** moves the **/etc/gated.conf+** file into place as **/etc/gated.conf**, retaining the older versions of the file as described above. **gdc** will decline to do anything when given this command if the new configuration file doesn't exist.

**backout**

**backout** rotates the configuration files in the newer direction, in effect moving the old configuration file to **/etc/gated.conf**. The command will decline to perform the operation if **/etc/gated.conf-** doesn't exist or is zero length, or if the operation would delete an existing, non-zero length **/etc/gated.conf+** file.

**BACKOUT**

**BACKOUT** performs a **backout** operation even if **/etc/gated.conf+** exists and is of non-zero length.

**modeconf**

**modeconf** sets all configuration files to mode **664**, owner **root**, group **gdmaint**. This allows a trusted non-root user to modify the configuration files.

**createconf**

If **/etc/gated.conf+** does not exist, **createconf** creates a zero length file with the file mode set to **664**, owner **root**, group **gdmaint**. This allows a trusted non-root user to install a new configuration file.

The following commands provide support for starting and stopping GateD and for determining its running state:

**running**

**running** determines whether GateD is currently running. This is done by checking to see whether GateD has a lock on the file containing its PID, whether the PID in the file is sensible, and whether there is a running process with that PID. **running** exits with zero status if GateD is running; non-zero otherwise.

**start**

**start** starts GateD. The command returns an error if GateD is already running. Otherwise **start** executes the GateD binary and waits for up to the delay interval (10 seconds by default, as set with the **-t** option otherwise) until the newly started process obtains a lock on its PID file. A non-zero exit status is returned if an error is detected while executing the binary, or if a lock is not obtained on its PID file within the specified wait time.

**stop**

**stop** stops GateD, gracefully if possible, ungracefully if not. The command returns an error (with non-zero exit status) if GateD is not currently running. Otherwise it sends a terminate signal to GateD and waits for up to the delay interval (10 seconds by default, as specified with the **-t** option otherwise) for the process to exit. Should GateD fail to exit within the delay interval, it is then signaled again with a second terminate signal. Should it fail to exit by the end of the second delay interval, it is signaled for a third time with a kill signal. This should force immediate termination. **stop** terminates with zero exit status when it detects that GateD has terminated, non-zero otherwise.

**restart**

If GateD is running, **restart** terminates it via the same procedure as is used for the **stop** command above. When the previous GateD terminates, or if it was not running prior to command execution, a new GateD process is executed using the procedures described for the **start** command above. A non-zero exit status is returned if any step in this procedure appears to have failed.

The following commands allow the removal of files created by the execution of some of the commands above:

**rmcore**

**rmcore** removes any existing GateD core dump file.

**rmdump**

**rmdump** removes any existing GateD state dump file.

**`rmparse`**

`rmparse` removes the parse error file generated when a **`checkconf`** or **`checknew`** command is executed and syntax errors are encountered in the configuration file being checked.

**`version`**

`version` shows the version information for GateD. GateD cannot already be running at the time **`version`** is executed. No options of **`gdc`** are used with this command.

## 3.5 Related Files

Many of the default filenames listed below contain the string %s, which is replaced by the name with which GateD is invoked. Normally this is **`gated`**, but if invoked as **`gated-test`**, GateD will by default look for **`/etc/gated-test.conf`**. These paths may all be changed at compilation time.

**`/etc/gated`**

**`/etc/gated`** is the location of the GateD binary. Another popular location is **`/usr/local/sbin/gated`**.

**`/etc/gated.conf`**

**`/etc/gated.conf`** is the location of the current GateD configuration file.

**`/etc/gated.conf+`**

**`/etc/gated.conf+`** is the location of the newer configuration file.

**`/etc/gated.conf-`**

**`/etc/gated.conf-`** is the location of an older configuration file.

**`/etc/gated.conf--`**

**`/etc/gated.conf--`** is the location of a much older configuration file.

**`/etc/gated.pid`**

**`/etc/gated.pid`** is the location where GateD stores its PID. The default is **`/etc/%s.pid`**. Another popular location is **`/var/run/%s.pid`**.

**`/var/tmp/gated_dump`**

**`/var/tmp/gated_dump`** is the location of GateD's state dump file. The default is **`/var/tmp/%s_dump`**. Another popular location is **`/usr/tmp/%s_dump`**.

**`/var/tmp/gated_parse`**

**`/var/tmp/gated_parse`** is where config file parse errors go. The default is **`/var/tmp/%s_parse`**. Another popular location is **`/usr/tmp/%s_parse`**.

**`/var/tmp`**

**`/var/tmp`** is where GateD drops its core file. Another popular location is **`/usr/tmp`**. The core file is usually **`core`**, but some systems use **`core.gated`**.

## 3.6 Bugs

Many commands work only when GateD is installed in the system directory with which it was configured.

There is not yet any way to tell **`gdc`** about systems that name their core dump other than **`core`** (**`core.gated`** is a less common possibility).

# Chapter 4
# OSPF Monitor (ospfmon)

OSPF Monitor is experimental.

## 4.1 Name

**`ospfmon`** - monitors OSPF gateways

## 4.2 Synopsis

```
ospfmon [-a] [-c] [-e] [-h] [-l] [-x] [-r] [-i] [-n] [-v] [-s] [-A area]
    [-C] [-I intf] [-L lsid] [-R advrtr] [-T type] [-V host]
```

## 4.3 Description

Use the **`ospfmon`** command to query OSPF routers. The **`ospfmon`** command operates in an interactive mode. It allows the user to query the various OSPF routers to provide detailed information about:

- IO statistics
- error logs
- link-state databases
- AS external databases
- the OSPF routing table
- configured OSPF interfaces
- OSPF neighbors

**Note:** This utility is provided for users of older versions of GateD and the old OSPF implementation. The new OSPF implementation implements many of the same features, but not all are available. Usage of this command is discouraged for new OSPF users; use the dump or GII facility instead.

## 4.4 Commands

The commands are specified using command-line flags. A description of each command follows.

**`-a`** - dump lsa (requires *`area`*, *`type`*, *`lsid`*, *`advrtr`*)

**`-c`** - show counters (default)

**`-C`** - don't show counters

**`-h`** - show next hops

**`-l`** - dump intra-area LSDB

**`-x`** - dump ASE LSDB

**`-r`** - show routes

**`-i`** - show interfaces

**`-n`** - show neighbors

**`-v`** - dump vertices

**`-V`** *`host`* - show software version

The following commands are used to specify additional parameters for the commands above. Each has a default value.

**`-A`** *`area`* - specify area, default is 0 (backbone)

**`-I`** *`intf`* - specify interface, default is IP address of queried host

**`-L`** *`lsid`* - LSID, default is IP address of queried host

**`-R`** *`advrtr`* - advertising router, default is IP address of queried host

**`-T`** *`type`* - type, default is 1 (Router LSA)

**`-K`** - authentication key, 8 bytes long, default is no auth

## 4.5 Sample Command Output

### 4.5.1 The ‘c’ command

The following is an example output of the '**c**' (show counts) command. This command shows a cumulative log of values pertaining to OSPF state.

```
IO stats
    Input  Output  Type
        5       0  Monitor request
      171      85  Hello
        7       8  DB Description
        2       5  Link-State Req
       80      14  Link-State Update
       56      56  Link-State Ack
    ASE: 100 checksum sum 36F1AC
    LSAs Originated: 25     Received: 156
            Router: 6  ASE: 19
    Area 0.0.0.2:
            Neighbors: 2    Interfaces: 1
            Spf: 4  Checksum sum 42FDC
            DB: rtr: 3 net: 1 sumasb: 5 sumnet: 2
```

```
Routing Table:
    Intra Area: 1   Inter Area: 2   ASE: 79
done
```

### 4.5.2 The ‘e’ command

The '**e**' command shows cumulative errors. Sample output from this command follows.

```
Packets Received:
   7: Monitor request       183: Hello
   7: DB Description            2: Link-State Req
  84: Link-State Update      58: Link-State Ack

Packets Sent:
   0: Monitor response        91: Hello
   8: DB Description              5: Link-State Req
  14: Link-State Update         58: Link-State Ack

Errors:
0: IP: Bad destination          0: IP: Bad protocol
0: IP: Received my own packet  0: OSPF: Bad packet type
0: OSPF: Bad version              0: OSPF: Bad checksum
0: OSPF: Bad area id               0: OSPF: Area mismatch
0: OSPF: Bad virtual link   0: OSPF: Bad authentication type
91: OSPF: Bad authentication key  0: OSPF: Packet too small

done
```

# Chapter 5
# ripquery

ripquery is experimental.

## 5.1 Name

**`ripquery`** - query RIP gateways

## 5.2 Synopsis

```
ripquery [ -1 ] [ -2 ] [ -[ a | 5 ] authkey ] [ -n ] [ -N dest[/mask] ]
    [ -p ] [ -r ] [ -v ] [ -w time ] gateway ...
```

## 5.3 Description

**`ripquery`** is used to request all routes known by a RIP gateway by sending a **`RIP REQUEST`** or a **`RIP POLL`** command. The routing information in any routing packets returned is displayed numerically and symbolically. **`ripquery`** is intended to be used as a tool for debugging gateways, not for network management.

**`ripquery`**, by default, uses the **`RIP POLL`** command, which is an undocumented extension to the RIP specification supported by **`routed`** on *SunOS 3.x* and later. The **`RIP POLL`** command is preferred over the **`RIP REQUEST`** command because it is not subject to split horizon and/or poisoned reverse. See “Chapter 11 Routing Information Protocol (RIP)” on page 39 of *Configuring GateD* for more information.

## 5.4 Options

**`-1`**

**`-1`** sends the query as a version 1 packet.

**`-2`**

**`-2`** sends the query as a version 2 packet (default).

**`-[ a | 5 ]`** *authkey*

**`-[ a | 5 ]`** *authkey* specifies the authentication password to use for queries. If **`-a`** is specified, an authentication type of SIMPLE will be used. If **`-5`** is specified, an authentication type of MD5 will be used. Otherwise, the default is an authentication type of NONE, which indicates that authentication fields in incoming packets will be displayed but not validated.

**`-n`**

**`-n`** prevents the address of the responding host from being looked up to determine the symbolic name.

**-N** *dest[/mask]*

**-N** *dest[/mask]* specifies that the query should be for the specified *dest/mask* instead of the complete routing table. The specification of the optional mask implies a version 2 query. Up to 23 requests about specific destinations can be included in one packet.

**-p**

**-p** (lowercase) uses the **RIP POLL** command to request information from the routing table. This is the default, but it is a proprietary extension supported only by some versions of routed and later versions of GateD. If there is no response to the **RIP POLL** command, the **RIP REQUEST** command is tried. GateD responds to a **RIP POLL** command with all the routes learned via RIP.

**-r**

**-r** uses the **RIP REQUEST** command to request information from the gateway's routing table. Unlike the **RIP POLL** command, all gateways should support the **RIP REQUEST**. If there is no response to the **RIP REQUEST** command, the **RIP POLL** command is tried. GateD responds to a **RIP REQUEST** command with all the routes announced out the specified interface.

**-v**

**-v** (lowercase) displays version information about **ripquery** before querying the gateways.

**-w** *time*

The default *time* is 5 seconds.

## 5.5 Bugs

Some versions of UNIX do not allow looking up the symbolic name of a subnet.

# Chapter 6
# rip6query

rip6query is experimental.

## 6.1 Name

**`rip6query`** - query RIPng gateways

## 6.2 Syntax

```
rip6query [ -d ] [ -n ] [ -r ] [ -v ] [ -w time ] [ -N dest[/mask] ]
    [ -I interface name ] hosts ...
```

## 6.3 Description

**`rip6query`** is used to request all routes known by a RIPng gateway by sending a **`RIPng REQUEST`** command. The routing information in any routing packets returned is displayed numerically and symbolically. **`rip6query`** is intended to be used as a tool for debugging gateways, not for network management.

## 6.4 Options

**`-d`**

**`-d`** specifies that debug output should be printed. These messages are printed to standard output.

**`-n`**

**`-n`** prevents the address of the responding host from being looked up to determine the symbolic name.

**`-N`** *`dest`*`[`*`/mask`*`]`

**`-N`** *`dest`*[*`/mask`*] specifies that the query should be for the specified *`dest/mask`* instead of the complete routing table.

**`-r`**

**`-r`** uses the **`RIPng REQUEST`** command to request information from the gateway's routing table. This is the default and is provided for backwards compatibility.

**`-v`**

**`-v`** (lowercase) displays version information about **`rip6query`** before querying the gateways.

**`-w`** *`time`*

The default *`time`* is 5 seconds.

## 6.5 Bugs

Some versions of UNIX do not allow looking up the symbolic name of a subnet.

# Chapter 7
# GateD Interactive Interface (GII)

## 7.1 Overview

The GateD Interactive Interface (GII) provides an interactive interface to a running GateD daemon. This interface can be used to query internal GateD variables. GII, which is implemented like any other protocol in GateD, accepts telnet connections to port 616 (C.f. [RFC 854] for a description of the telnet protocol). After user identification, GII answers any query sent as ASCII commands. The commands include queries about the memory, routing table, interface list, and other internal parameters.

## 7.2 Configuration

GII will identify the user by using the UNIX password of the GII ID of the system. In other words, when using the GII interface, one must create a GII account by editing the **`/etc/passwd`** file.

Alternatively, any user name can be used by changing GII_USER in the **`gii.h`** file.

## 7.3 Connecting to GII

1. Open a telnet connection to the machine running GateD on TCP port 616.
2. Identify yourself using a simple password.
3. Enter **`quit`** to end your telnet session.

Here is an example of such a session:

```
% telnet router1.foo.bar 616

Trying 192.168.10.1...

Connected to router1.foo.bar.

Password?

100 GateD Interactive Interface. Version 4-0-6

GateD-router1.foo.bar> help

100 HELP: The possible commands are:

100     help: Print help messages

100     show: Show internal values

100     quit: Close the session

GateD-router1.foo.bar> show

100 HELP: The possible subcommands are:
```

```
100     version: Show the current GateD version
100     kernel: Show the Kernel support
100     interface [name|index]: Show interface status
100     memory: Show the memory allocation
100     ip: Show info about IP protocol
100     task: Show list of active tasks
100     dvmrp: Show info about DVMRP protocol
100     ospf: Show info about OSPF protocol
100     timer: Show list of timers
100     bgp: Show info about BGP protocol
100     rip: Show info about RIP protocol
100     ripng: Show info about RIPng protocol
GateD-router1.foo.bar> quit
```

## 7.4 Show Command Parameters

**show version**
**show version** displays the current GateD version.

**show kernel**
**show kernel** displays the kernel support.

**show interface [** *name* **|** *index* **]**
**show interface** displays the interface status on the interface specified.

**show memory**
**show memory** displays the allocation of memory blocks used by GateD.

**show ip [ route | walkup | walkdown | rpf ]** *prefix/len*
**show ip** displays the IPv4 routes in the GateD routing table.

**show task**
**show task** displays a list of active tasks.

**show ip mroute [ boundaries | mfc | static ]**
**show ip mroute** displays multicast routing information based on administrative boundaries, multicast forwarding cache or static routes.

**show ospf [ global | interface** *address* **| area** *area-ID* **[LS_type] ]**
Only OSPF version 2 is supported.

**show timer**
**timer** displays a list of timers.

**show ip igmp groups**
**show ip igmp groups** displays directly connected groups learned via IGMP.

**show bgp [ aspath [ regexp ] | cidr-only [** *prefix/masklen***] | community** *community number* **| peeras** *as_no* **| peer-group [ internal | external | internal_igp | routing | test ] | routes** *prefix/masklen* **| summary ]**
**show bgp** displays information about BGP routes. See page 24, “BGP”, for details on the BGP commands.

**show rip [ routes [** *prefix/masklen* **] | summary | tag [***tag***] ]**

**show rip** displays information about RIP routes.

## 7.5 GateD Core

The following commands will display parameters related to the internal core of GateD:

**show version**

**show version** displays the version of the running GateD.

**show kernel**

**show kernel** displays the type of kernel of the host, including which features are supported (for example, reject routes, Multicast, UDP checksums, and so on).

**show memory**

**show memory** displays the memory usage, divided by memory block structures.

**show task**

**show task** displays the running tasks in GateD.

**show timers**

**show timers** displays a list of timers.

## 7.6 Interfaces

The following command will display parameters related to the interfaces:

**show interface [** *name* **|** *index* **]**

Without a parameter, this command lists all the interfaces of the system. If one interface name or index is given as an argument, all the parameters concerning this interface are displayed.

## 7.7 Routing tables

The following commands will display parameters related to the routing table:

**show ip route [** *prefix/len* **]**

Without a parameter, **show ip route** prints the size of the IP routing table. With a parameter, which must be a prefix number and mask length, **show ip route** displays complete information about the given route, such as the number of announcements, next hop, AS path, active route, and so on.

**show ip walkup [** *prefix/len* **]**

**show ip walkup** lists all the routes that are less specific than *prefix/len,* for example, all the components of the aggregate *prefix/len*.

**show ip walkdown [** *prefix/len* **]**

**show ip walkdown** lists all the routes that are more specific than *prefix/len,* for example, all the components of the aggregate *prefix/len*. **show ip walkdown 0/0** will display the whole routing table. **^C** can be used to stop the listing.

**show ip6 walkup [** *prefix/len* **]**

**show ip6 walkup** lists all the routes that are less specific than *prefix/len.* For IPv6, for example, all the components of the aggregate *prefix/len*.

**show ip6 walkdown [** *prefix/len***]**

**show ip6 walkdown** lists all the routes that are more specific than *prefix/len*. For IPv6, for example, all the components of the aggregate prefix/len. **show ip walkdown 0/0** will display the whole routing table. **^C** can be used to stop the listing.

**show ip rpf [** *prefix/len* **]**

**show ip rpf [** *prefix/len* **]** displays the RPF information.

## 7.8 RIP

**show rip routes [** *prefix/masklen* **]**

**show rip routes** displays RIP information about routes.

**show rip summary**

**show rip summary** displays a table of information about all RIP routes.

**show rip tag** [ *tag* ]

**show rip tag** displays RIP tag information.

## 7.9 OSPF

**show ospf global**

**show ospf global** displays general OSPF options.

**show ospf interface address**

**show ospf interface address** displays IP interface status.

**show ospf area** *area-ID* **[ LS_TYPE ]**

**show ospf area** *area-ID* displays area information.

**show ospf asex**

**show ospf asex** displays the contents of the router-global ASEX (type-5) LSA Database.

**show ospf asopaque**

**show ospf asopaque** displays the contents of the router-global AS-Scope Opaque LSA (type-11) Database.

## 7.10 ISIS

**show isis global**

**show isis global** displays some global ISIS parameters

**show isis interface** *address*

**show isis interface** *address* displays the interface status.

## 7.11 BGP

**show bgp aspath [** *regexp* **]**

**show bgp aspath** displays **bgp aspath**'s that match the regular expression. If null, all paths should be sent.

**show bgp cidr-only [** *prefix/masklen* **]**

**show bgp cidr-only** prints only CIDR routes based the route table. Class prefixes of /8 , /16, and /24 are ignored. If no routes are specified, the tree walks from the top of the route table. If not, it walks from the **prefix/masklen**.

**show bgp community [** *community number* **]**

**show bgp community** displays BGP routes associated with the community number specified. The community number is specified as a hexadecimal number.

**show bgp peeras** *as_num*

**show bgp peeras** *as_num* displays BGP peer information associated with this AS, such as **bgp peer address**, **bgp versions**, **gateway** (third party). *as_num* is the AS number of the peers you want to track. An *as_num* must be specified or an error will result.

**show bgp peer-group [ internal | external | internal_igp | routing | test ]**

**bgp** displays summary information about all BGP peers in grouping, such as:
**peer info** - AS, BGP version, neighbor addresses
**statistics** - number of updates in/out, state

**show bgp routes** *prefix/masklen*

**show bgp** displays BGP information about these routes.

**show bgp summary**

**show bgp summary** displays summary information about all BGP peers, such as:
**peer info** - AS, BGP version, neighbor addresses
**statistics** - number of updates in/out, state

## 7.12 IGMP

**show ip igmp groups** displays directly connected groups learned via IGMP.

## 7.13 DVMRP

**show ip dvmrp interfaces** shows interfaces configured for DVMRP.

**show ip dvmrp mfc** shows DVMRP forwarding entries.

**show ip dvmrp neighbors** shows DVMRP neighbor information.

**show ip dvmrp errors** shows DVMRP RPF lookup errors.

**show ip dvmrp route ip_address** shows the DVMRP route for ip_address.

**show ip dvmrp routes [network][/network-mask]** shows all routes learned via DVMRP or the route given by **[network][/network-mask].**

**show ip dvmrp targets** shows information about DVMRP targets.

**show ip dvmrp df[network/netmask]** shows the DVMRP designated forwarder for **[network/netmask]** on all downstream interfaces.

**show ip dvmrp prunes** shows DVMRP prunes received.

**show ip dvmrp prunexmits** shows DVMRP prunes sent.

## 7.14 PIM-SM

**show ip pim bsr** displays BSR status.

**show ip pim crp** displays the crp status.

**show ip pim interface** displays pim interface information.

**show ip pim neighbor** displays PIM-SM neighbor information.

**show ip pim route** displays PIM-SM routing tables.

**show ip pim rp-hash ip-address** displays the group hash mapping for ip-address.

**show ip pim rpset** displays the RendezvousPoint-Set.

**show ip pim timeouts** displays all component timeouts.

**show ip pim walkup** *prefix/masklen* displays less specific routes for multicast.

**show ip pim walkdown** *prefix/masklen* displays more specific routes for multicast.

## 7.15 Multicast Routing Table

**show ip mroute boundaries**

**show ip mroute boundaries** displays administratively-scoped boundary information.

**show ip mroute mfc**

**show ip mroute mfc** displays forwarding cache information.

**show ip mroute static**

**show ip mroute static** displays static group membership information.

## 7.16 Miscellaneous GII Commands

**gii-clear-interface-times** *machine* clears the interface timers that record the first and last interface change.

**gii-diff-interface-times** *machine* shows the time elapsed between first learned and last learned interfaces

**gii-diff-routes-learned-time** *machine first_route/masklen last_route/masklen* shows the time elapsed between two learned routes. The routes should be in the format route/masklength.

**gii-diff-routes-update-time** *machine first_route/masklen last_route/masklen* shows the time elapsed between two routes that were updated. The routes should be in the format route/masklength.

**gii-get-route-learned-time** *machine* **[route]** displays the time a route was learned. If no route was specified, a list of all routes in GateD will be displayed.

**gii-show-interface-times** *machine* shows interface times. The first interface time is the first learned interface, and the second interface time is the last learned interface.

## 7.17 Interface Commands

**add-multiple-aliases [amount]** adds simultaneously to each machine specified by the set-machines command aliases with the subnet of 10.128/9. Rack number and machine letter is used to obtain unique addresses.

**add-multiple-aliases** *machine if_name start_addr num_aliases* **[offset]** adds a sequence of aliases on machine *machine* and associates the addresses with a specific interface *if_name*. The *start_addr* represents the address. If no mask was specified, a mask of length 24 is used as the default value.

If this command is used in tunnel sequence settings, the keyword **other** can be used to specify the other side of the tunnel. In this case the addresses are offset by 0.1.0.0 from the specified *start_addr*.

**add-multiple-tunnels** *machine if_name start_addr num_tunnels* **[option] [offset]** adds a sequence of tunnels on machine *machine* and associates the addresses with a specific interface *if_name*. The *start_addr* represents the address of the alias to associate with the tunnel. All tunnel addresses are offset from this address by 0.10.0.0. This command associates the tunnel with the alias address, unless otherwise specified by the **`single`** keyword, which will then read the interface IP address and associate the tunnel to it.

To set the other side of the tunnel, the **`other`** keyword is used without changing the **`start_addr`** of the first side.

If x gif tunnels already exist, you can specify the offset number to leave the first **`[offset]`** gifs intact.

**`add-multiple-tunnels-verbose`** *`machine if_name start_addr num_tunnels`* **`[option] [offset]`** is equivalent to **`add-multiple-tunnels`** except that it displays additional information about time and memory values.

**`flap-tunnels`** *`interface starting_address amount count`* **`[offset]`** will flap tunnels by adding *`amount`* gif interfaces and subsequently removing the interfaces *`count`* number of times. **`[offset]`** again specifies here whether the tunnels will be created from a given gif address. These tunnels cannot be used for routing since they set only one side of the machine and use the IP address of *`interface`* to associate the tunnels to.

**`updown-tunnels`** *`machine interface starting_address amount count`* **`[offset]`** is similar to the **`flap-tunnels`** command with the difference that tunnels are not created, but the gif interfaces are set up and down.

**`remove-multiple-aliases`** *`machine if_name start_addr num_tunnels`* **`[option] [offset]`** will remove *`num_tunnels`* aliases sequentially with addresses starting at *`start_addr`* on interface *`if_name`*.

**`remove-multiple-tunnels`** *`machine if_name start_addr num_tunnels`* **`[offset]`** will remove *`num_tunnels`* tunnels sequentially starting at **`[offset]`**, which represents the gif number. If no offset is specified the default value '0' is used. Although the start address and interface are required by the command, it isn't really used and is left here for consistent syntax with the rest of the tunnel commands.

## 7.18 Status Commands

**`compare-tunnels [machines]`** is similar to standard compare function, but differs in that in addition to comparing routes, it also compares tunnels. It does not have the capability to compare ribs.

**`compare [machines]`** runs compare results on **`[machines]`**.

# Chapter 8
# gdc dump

## 8.1 Purpose of a GateD Dump File

The purpose of a GateD dump file is to create an ASCII representation of GateD's internal state. A GateD dump file is not a core file and cannot be used with a debugger.

Most protocols and pseudo-protocols in GateD implement a "dump method" that is called to dump data specific to the module. In addition, the entire routing table is dumped (including an ASCII representation of the radix trie(s)) along with other pieces of core state. This data is a "snapshot" of the internal state at some point in time. Due to resource scheduling, this may not be the exact moment at which the dump signal was received.

## 8.2 When to Create a Dump File

Create a dump if GateD is running with a known or suspected error.

The GII and SNMP tools provide alternatives to a GateD dump. GII provides protocol info, a view of the current state of GateD, and a routing table listing.

## 8.3 How to Create a Dump

Create a GateD dump file using one of the following:

- **`gdc`** dump command
- **`SIGINT`** signal from UNIX system

To create a dump using the dump command, type

```
gdc dump
```

To create a dump with **`SIGINT`**, send a **`SIGINT`** signal to GateD. GateD forks a child process to perform the dump while the parent continues on as before. The kill(1) program is typically used to send signals to processes.

## 8.4 Finding a Dump

The default file location for a GateD dump is **`/var/tmp/gated.dump`**. This path is determined at compile time; the alternative location is **`/usr/tmp/gated_dump`**.

## 8.5 Format of GateD Dump File

GateD dumps include information about the following:

- general GateD state
- tasks
- memory usage and statistics

- interfaces
- martians
- routing table(s) (including info about configured statics)
- policy
- protocol-specific data

### 8.5.1 General Information in Dump

The first section of the dump will contain general information such as:

- GateD version
- date of dump
- operating system
- tracing flags set

In the following example,

- the GateD version is 8.0.0
- the date the dump was generated is January 18, 2000
- the operating system used is *SunOS 5.6*
- the tracing flags set are **`all`**
  - the file is open
  - 12 tasks reference the file
  - the current logfile size is 88,542 bytes
  - the maximum size is 102,4000 bytes
  - 2 old versions of the file are kept

```
  gated[12843] version 8-0-0 memory dump on paris.merit.edu at Tue Jan 18
01:46:41 2000


  SunOS 5.6 Generic_105181-17 paris.merit.edu sun4u sparc


  Started at Tue Jan 18 01:42:30 2000


  Tracing:
      Global: all
      Files:
          /var/tmp/gated.log:
  opened  refcount 12  size 88542  limit 1024000  files 3
```

### 8.5.2 Task State

The task section of the dump contains information about the task state, such as:

- send/recv buffer size
- task name

- task priority
- tracing options for task
- trace file for task
- timers associated with task

In the following example:

- the send buffer size is 16384, and the receive buffer size is 8,192 bytes
- the task names are IF, INET, Aggregate, RT, ICMP, and BGP.0.0.0.0+179
- the task priorities for each task are, respectively, 10, 15, 20, 20, 30, and 40
- the tracing options are: general, state, policy, task, and timer for all names except BGP, which has only packets
- the trace files are all : /var/tmp/gated.log
- the only timer is associated with IF and is IF_AGE
    - the time of last firing is 01:45:30.259 am; next firing is at 01:48:30.259 AM

---

```
Task State: <>
     Send buffer size 16384 at 150000
     Recv buffer size 8192 at 14A000


Tasks (12) and Timers:
IF              Priority 10     RtProto Direct
                Trace options: general state policy task timer
                Trace file: /var/tmp/gated.log  size 1024000 files 3

                IF_AGE  <oneshot>
                last: 01:45:30.259      next: 01:48:30.259

INET            Priority 15     Socket  4       RtProto INET
                Trace options: general state policy task timer
                Trace file: /var/tmp/gated.log  size 1024000 files 3

Aggregate       Priority 20     RtBit 1
                Trace options: general state policy task timer
                Trace file: /var/tmp/gated.log  size 1024000 files 3

RT              Priority 20
                Trace options: general state policy task timer
                Trace file: /var/tmp/gated.log  size 1024000 files 3

ICMP            Priority 30     Proto   1       Socket  6
                Trace options: general state policy task timer
```

```
BGP.0.0.0.0+179 Priority 40     Port   179      Socket  8    RtProto BGP
                Trace options: packets
                Trace file: /var/tmp/gated.log  size 1024000 files 3
```

### 8.5.3 Socket Read Routines

The next section of the dump will contain socket read information such as:

- read routines (in the example: KRT, ICMP, GII_LISTEN)
- low priority read routines (BGP.0.0.0.0+179)
- file descriptors (files and sockets)

```
  Socket read routines:

     1       KRT
     6       ICMP
     7       GII_LISTEN.0.0.0.0+616 (accept)

  Socket low priority read routines:

     8       BGP.0.0.0.0+179 (accept)

     File Descriptors (max 8):
             0
             1       Task: KRT        read rqueue
             2       Task: KRT       File: krt_ifread_task
             3       File: /etc/gated.pid
             4       Task: INET
             5
             6       Task: ICMP       read rqueue
             7       Task: GII_LISTEN.0.0.0.0+616    accept rqueue
             8       Task: BGP.0.0.0.0+179           accept
```

### 8.5.4 Memory Usage in Dump

This section of the dump will contain memory usage information for each task block such as:

- allocation size (8,192 bytes in the example)
- number of free blocks (935 blocks)
- the last block allocated (001483BO)
- data types for which memory is allocated (runt, krt_remnant_rt, vtxlist_t, nospf_route_link_t)

```
Task Blocks:
Allocation size:  8192


Size: 8 N_free: 935 LastAlloc: 001483B0
runt                Init:     5  Alloc:      0  Free:      0  InUse:     0
krt_remnant_rt      Init:     1  Alloc:      5  Free:      5  InUse:     0
vtxlist_t           Init:     1  Alloc:      0  Free:      0  InUse:     0
nospf_route_link_t  Init:     1  Alloc:      0  Free:      0  InUse:     0
iflist_t            Init:     1  Alloc:      0  Free:      0  InUse:     0
sockaddr_un.in      Init:     1  Alloc:    309  Free:    217  InUse:    92
asmatch_t           Init:     1  Alloc:      0  Free:      0  InUse:     0
```

### 8.5.5 Interface - Task and Logical Addresses

The interface section of the dump will contain interface information for each task such as:

- physical interfaces assigned and up (in the example, IF has 2 physical interfaces assigned and 2 up)
- INET protocol addresses assigned and up (2 assigned and 2 up)
- local and physical addresses to which the interface connects (local: 120.0.0.1, 198.108.60, 192.168.10, and 0.0.0.142; physical: 8:0:20:a8:8c:60 )
- interface names (lo, lo0, hme, hme0, hme1)
  - index number (for lo:1)
  - change (for lo:<> )
  - state (for lo: `<up loopback multicast>`)
  - refcount (for lo:2)
  - up-down transitions (for lo:0 120.0.1)
  - metric (for 120.0.1:0)
  - MTU (for 120.0.1:1472 )
  - refcount (for 120.0.1:3)
  - preference when interface is up (for 120.0.1:0)
  - preference when interface is down (for 120.0.1:120)

- change (for 120.0.1: <> )
- state (for 120.0.1: `<up loopback multicast>`)
- subnet mask (for 120.0.1: 255.255.255)

---

```
Task IF:
     Physical interfaces: 2          Up: 2
     INET protocol addresses: 2      Up: 2

Addresses:
     120.0.1
     P2P 0   Loop 1  Total 1  Refcount 2  Route: not installed
     198.108.60.142
     P2P 0   Loop 0  Total 1  Refcount 1  Route: not installed
     192.168.10.142
     P2P 0   Loop 0  Total 1  Refcount 1  Route: not installed

Local addresses:
     127.0.0.1
     P2P 0   Loop 1  Total 1  Refcount 1  Route: not installed
     198.108.60.142
     P2P 0   Loop 0  Total 1  Refcount 1  Route: not installed
     192.168.10.142
     P2P 0   Loop 0  Total 1  Refcount 1  Route: not installed
     0.0.0.142
     P2P 0   Loop 0  Total 0  Refcount 2  Route: not installed

Physical addresses:
    802.2 8:0:20:a8:8c:60
    Refcount 2
    Names:
     lo
             Refcount 1
     lo0
             Refcount 1
     hme
             Refcount 2
     hme0
             Refcount 1
     hme1
             Refcount 1
    Interfaces:
        lo0     Index 1         Change: <>    State:<up loopback multicast>
             Refcount: 2     Up-down transitions: 0

             120.0.1
                     Metric: 0       MTU: 1472
                     Refcount: 3     Preference: 0   Down: 120
                     Change: <>      State:  <up loopback multicast>
                     Subnet Mask: 255.255.255.255

        hme0    Index 2 Address 802.2 8:0:20:a8:8c:60   Change: <>   State:
             Refcount: 2     Up-down transitions: 0

             198.108.60.142
                     Metric: 0       MTU: 1436
                     Refcount: 3     Preference: 0   Down: 120
                     Change: <>      State:  <up broadcast multicast>
                     Broadcast Address:   198.108.60.255
                     Subnet Number: 198.108.60    Subnet Mask: 255.255.255

        hme1    Index 3 Address 802.2 8:0:20:a8:8c:60    Change: <>   State
:<up broadcast multicast>
             Refcount: 2     Up-down transitions: 0

             192.168.10.142
                     Metric: 0       MTU: 1436
                     Refcount: 16    Preference: 0   Down: 120
                     Change: <>      State: <up broadcast multicast>
                     Broadcast Address:   192.168.10.255
                     Subnet Number: 192.168.10    Subnet Mask: 255.255.255
```

### 8.5.6 INET - Martians

The martians section of the dump will contain INET task information, which is where the martians configuration is stored. Martians are configured using the '**`martians`**' clause. (See "Martian Syntax" on page 30 in *Configuring GateD* for more information.) The dumped data includes the state of:

- IP forwarding (in the example, INET has 0 IP forwarding)
- UDP checksums (1)
- reject address (127.0.0.1)
- blackhole address (127.0.0.1)
- autonomous system (201)
- router ID (0.0.0.142)
- martians (0.0.0.0, 127, 255.255.240)

```
  Task INET:
     IP forwarding: 0             UDP checksums: 1

     Reject address: 127.0.0.1    Blackhole address: 127.0.0.1

     Autonomous system: 201
     Router ID: 0.0.0.142

     Martians:
             0.0.0.0          mask 0.0.0.0         Exact
             0.0.0.0          mask 255
             127              mask 255             restrict
             255.255.240      mask 255.255.240     restrict
```

### 8.5.7 Route Table (RT) Information

This section of the dump contains information about configured static routes as well as the state of the RIBs (Routing Information Base(s)) for the various address families. Only 'inet' will be shown in the following examples. The routes are printed in an ASCII representation of the radix trie as well as in a detailed listing on a per-destination basis.

```
Task RT:
     Static routes for family INET: (* indicates gateway(s) in use)
             Radix trie for inet (2) inodes 27 routes 15:

                                 +-24+--{223.145.154
                              +-22+
                              |  |  +-24+--{223.145.153
```

```
         |  +-23+
         |     +-24+--{223.145.152
      +-20+
      |  |        +-24+--{223.145.151
      |  |     +-23+
      |  |     |  +-24+--{223.145.150
      |  |  +-22+
      |  |  |  |  +-24+--{223.145.149
      |  |  |  +-23+
      |  |  |     +-24+--{223.145.148
      |  +-21+
      |     |     +-24+--{223.145.147
      |     |  +-23+
      |     |  |  +-24+--{223.145.146
      |     +-22+
      |        +-24+--{223.145.145
   +--8+
   |  |     +-24+--{223.3
   |  |  +-15+
   |  |  |  +-24+--{223.2
   |  +-14+
   |     +-24+--{223.1
+--0+--{0.0.0.0
|  +--8+--{127
```

It also includes static route detail information, such as:

- net mask set for the route (in the example, mask 0.0.0.0 is set for route 0.0.0.0; mask 255 is set for route 127)
- preference set for the route (preference 60 for route 0.0.0.0; 0 for route 127)
- which gateway/interface is used for the route (gateway 198.108.60.1 is set for route 0.0.0.0;
- interfaces 127.0.0.1 is set for route 127)

```
0.0.0.0   mask 0.0.0.0     preference 60  state <int retain gateway>  Gate-
way  198.108.60.1(*)


127      mask 255        preference 0   state <noadvise int retain reject>
Interfaces 127.0.0.1
```

```
223.1     mask 255.255.255 preference 60  state   Gateway 192.168.10.1(*)


223.2     mask 255.255.255 preference 60  state <int gateway>  Gateway
192.168.10.1(*)


223.3     mask 255.255.255 preference 60  state <int gateway>  Gateway
192.168.10.1(*)
```

The next portion of the dump shows static gateway information, such as which gateways and tasks are referenced by static routes. (In this example, no routes reference either of the gateways. Task RT is referenced by both of the routes.)

```
Gateways referenced by static routes:
              198.108.60.1
                     routes: 0 task: RT
              192.168.10.1
                     routes: 0 task: RT
```

This is followed by static route announce bit information, such as bit allocations, as shown in the example below.

```
Bit allocations:
     1       Aggregate
     2       KRT     byte index: 0   length: 4
```

Static route mask and address information are shown in the example below, such as:

- family
- address
- length
- mask

```
Masks and addresses:


             Family  Address Length  Mask
             inet    1317B0  0       0.0.0.0
             inet    1317B8  1       128
             inet    1317C0  2       192
             inet    1317C8  3       224
             inet    1317D0  4       240
             inet    1317D8  5       248
             inet    1317E0  6       252
             inet    1317E8  7       254
```

---

The sixth section of the dump will also include a graph of the real (nonstatic) routes as shown in the example below.

---

```
Routing Tables:
     Radix trie for inet (2) inodes 32 routes 18:

                            +-24+--{223.145.154
                         +-22+
                         |  |  +-24+--{223.145.153
                         |  +-23+
                         |     +-24+--{223.145.152
                      +-20+
                      |  |        +-24+--{223.145.151
                      |  |     +-23+
                      |  |     |  +-24+--{223.145.150
                      |  |  +-22+
                      |  |  |  |  +-24+--{223.145.149
                      |  |  |  +-23+
                      |  |  |     +-24+--{223.145.148
                      |  +-21+
                      |     |     +-24+--{223.145.147
                      |     |  +-23+
                      |     |  |  +-24+--{223.145.146
                      |     +-22+
                      |        +-24+--{223.145.145
                   +--8+
                   |  |     +-24+--{223.3
                   |  |  +-15+
                   |  |  |  +-24+--{223.2
                   |  +-14+
                   |     +-24+--{223.1
                +--3+
                |  |  +-24+--{198.108.60
                |  +--5+
                  |     +-24+--{192.168.10
               +--0+--{0.0.0.0
               |  +--8+--{127
               |     +-32+--{127.0.0.1



       + = Active Route, - = Last Active, * = Both
```

The next section of the dump shows real route detail information, such as:

- for all the routes:
  - number of destinations (in the example that follows, 18 destinations exist for these routes)
  - number of routes (18)
  - number of holddowns (0)
  - number of deleted routes (10)
  - number of hidden routes (0)
- for each route:
  - net mask (for route 0.0.0.0, the mask is 0.0.0.0)
  - number of entries (for route 0.0.0.0, the number of entries is 1)
  - announce (1)
  - TSI (none)
  - instability histories (none)
  - *static preference (60)
    **Note**: The asterisk (*) indicates that route is active.
  - next hop (198.108.60.1)
  - interface (198.108.60.142(hme0))
  - state bits set<int active retain gateway>
  - age (4:11)
  - metric (0)
  - metric2 (0)
  - tag (0)
  - task (RT)
  - announcement bits(1) (2-KRT)
  - AS path (IGP (Id 1))

```
Routing table for inet (2):
           Destinations: 18        Routes: 18
           Holddown: 0     Delete: 10      Hidden: 0

   0.0.0.0     mask 0.0.0.0
                   entries 1       announce 1
                   TSI:
                   Instability Histories:

           *Static  Preference:  60
                   NextHop: 198.108.60.1  Interface: 198.108.60.142(hme0)
                   State: <int active retain gateway>
                   Age: 4:11    Metric: 0    Metric2: 0    Tag: 0
                   Task: RT
                   Announcement bits(1): 2-KRT
                   AS Path: IGP (Id 1)


   127         mask 255
                   entries 1       announce 1
                   TSI:
                   Instability Histories:

           *Static  Preference:   0
                   NextHop: 127.0.0.1  Interface: 127.0.0.1(lo0)
                   State: <noadvise int active retain reject>
                   Age: 4:11    Metric: 0    Metric2: 0    Tag: 0
                   Task: RT
                   Announcement bits(1): 2-KRT
                   AS Path: IGP (Id 1)


   127.0.0.1   mask 255.255.255.255
```

## 8.5.8 Example Protocol-Specific Data: BGP Peers and Groups

The protocol-specific section of the dump includes information about all protocols that implement a dump method. The example shown here shows BGP peer and group information, such as:

- task name
  - task groups
  - task total peers
  - task active incoming peers
  - task free peers
  - task free groups
  - task state
  - task autonomous system (AS) number
  - import controls for each AS
  - net mask for each AS

```
Task BGP.0.0.0.0+179:
Groups: 1    Peers: 1 (1 configured)  Active Incoming: 0
Free Peers: 0Free Groups: 0           State: Listening
     AS 202:
             Import controls:
                     0.0.0.0          mask 0.0.0.0


     AS 201:
             Import controls:
                     0.0.0.0          mask 0.0.0.0


     AS 202:
             Export controls:
                     Protocol BGP  as 202
                             0.0.0.0          mask 0.0.0.0
                     Protocol BGP  as 201
                             0.0.0.0          mask 0.0.0.0
                     Protocol Static
                             0.0.0.0          mask 0.0.0.0


     AS 201:
             Export controls:
                     Protocol BGP  as 202
```

```
                              0.0.0.0          mask 0.0.0.0
                      Protocol BGP  as 201
                              0.0.0.0          mask 0.0.0.0
                      Protocol Static
                              0.0.0.0          mask 0.0.0.0



Task BGP_202.192.168.10.101+179:
     Peer: 192.168.10.101+179   Local: 192.168.10.142  Type: External
     State: Active   Flags: <>
     Last State: Idle           Last Event: Start      Last Error: None
     Options: <>

Task BGP_Group_202_201:
     Group Type: External    AS: 202  Local AS: 201  Flags: <>
     Total Peers: 1          Established Peers:
```

---

Another, more complete example of a BGP dump file follows:

---

```
        BGP.0.0.0.0+179 Priority 40     Port   179      Socket 11
RtProto
BGP
                Trace options: packets
                Trace file: /var/tmp/gated.log  size 1024000    files 3

        BGP_201.192.168.10.50+33073     Priority 50     Port 33073
Socket 1
2       RtProto BGP     RtBit 3
                Trace options: packets
                Trace file: /var/tmp/gated.log  size 1024000    files 3

                BGP_201.192.168.10.50+33073_Traffic
                        last: 14:47:13.130      next: 14:47:13.130

        BGP_Group_201_202       Priority 50     RtProto BGP
                Trace options: packets
                Trace file: /var/tmp/gated.log  size 1024000    files 3
        ASMatch Priority 50

        ASPaths Priority 50

<.....>

        223.239         mask 255.255.255
                        entries 1       announce 1
                        TSI:
                        Instability Histories:

                *BGP    Preference: 170         Source: 192.168.10.50
                        NextHop: 192.168.10.254         Interface:
192.168.10.32
(hme1)
                        State:
                        Local AS:   202 Peer AS:   201
                        Age: 10 Metric: -1      Metric2: -1     Tag: 0
                        Task: BGP_201.192.168.10.50+33073
                        Announcement bits(1): 2-KRT
                        AS Path: (202) {201}IGP (Id 6)

        223.249         mask 255.255.255
                        entries 1       announce 1
                        TSI:
                             BGP 192.168.10.50 (External AS 201) no metrics
                        Instability Histories:

                *Static Preference:  60
                        NextHop: 192.168.10.250         Interface:
192.168.10.32
(hme1)
                        State:
                        Age: 7  Metric: 0       Metric2: 0      Tag: 0
                        Task: RT
                        Announcement bits(2): 2-KRT 3-
BGP_201.192.168.10.50+3307
3
```

```
                         AS Path: IGP (Id 1)

<......>

Task BGP.0.0.0.0+179:
        Groups: 1       Peers: 1 (1 configured) Active Incoming: 0
        Free Peers: 0   Free Groups: 0          State: Listening
        AS Path: AS Path Regular Expression: (.*)

<......>

Task BGP_201.192.168.10.50+33073:
       Peer: 192.168.10.50+33073      Local: 192.168.10.32+179       Type:
External
        State: Established      Flags: <>
        Last State: OpenConfirm Last Event: RecvKeepAlive       Last Error:
None
        Options: <>
        Peer Version: 4 Peer ID: 0.0.0.50       Local ID: 0.0.0.5
Active Holdtime: 180
        Last traffic (seconds): Received 10     Sent 22 Checked 27
        Input messages: Total 7 Updates 5       Octets 270
        Output messages:        Total 7 Updates 4       Octets 244
        Route Queue Timer: unset        Route Queue: empty

Task BGP_Group_201_202:
        Group Type: External    AS: 201 Local AS: 202   Flags: <>
        Total Peers: 1          Established Peers: 1

Task ASMatch:

Task ASPaths:
   Id 1    Refs 7 Hash 0   Lengths: Path 0, Seg 0, Attr 0, Aggr 0, Alloc'd 96
                Path: IGP
   Id 2    Refs 1 Hash 75  Lengths: Path 2, Seg 2, Attr 0, Aggr 0, Alloc'd 96
                Path: (202) 201 IGP
   Id 4    Refs 1 Hash 75  Lengths: Path 2, Seg 2, Attr 0, Aggr 0, Alloc'd 96
                Path: (202) 201 IGP
   Id 6    Refs 1 Hash 75  Lengths: Path 2, Seg 2, Attr 0, Aggr 0, Alloc'd 96
```

```
                Path: (202) 201 IGP
```

---

## 8.5.9 AS Paths

The next section of the dump will include AS path information for each ID, such as:

- references (for ID1, 18 references exist)
- hash (0)
- lengths (Path 0, Seg 0, Attr 0, Aggr 0, Alloc'd 100)
- path (IGP)

```
Task ASPaths:
Id 1Refs 18 Hash 0  Lengths: Path 0, Seg 0, Attr 0, Aggr 0, Alloc'd 100
   Path: IGP
```

---

## 8.5.10 Kernel

The final section of the dump will include kernel information, such as:

- options: <>
- kernel support: <reject blackhole varmask host>
- remnant timeout: 3:00
- routes in kernel: 18
- limit: unlimited
- flash install limit: 20
- flash install routes: interface
- background install limit: 120
- background install routes: low
- kernel install state: normal
- interface add/delete/change queue: 0
- queued High priority deletion queue: 0
- queued High priority change queue: 0
- queued High priority add queue: 0
- queued Normal priority deletion queue: 0
- queued Normal priority change queue: 0
- queued Normal priority add queue: 0 queued

```
Task KRT:
   Options: <>     Kernel support: <reject blackhole varmask host>
   Remnant timeout: 3:00
   Routes in kernel: 18            Limit: unlimited
   Flash install limit: 20         Flash install routes: interface
   Background install limit: 120   Background install routes: low
```

```
Kernel install state: normal

Interface add/delete/change queue: 0 queued
High priority deletion queue: 0 queued
High priority change queue: 0 queued
High priority add queue: 0 queued
Normal priority deletion queue: 0 queued
Normal priority change queue: 0 queued
Normal priority add queue: 0 queued
```

## 8.6 Dump vs. GII

The GII and dump facilities both have route table and protocol information. The dump is a more detailed listing of the internal state, while GII tends to be easier to read for quick status information. The GII interface can also be easier to use for those familiar with other vendors' command-line interfaces. The dump file should be emailed to NextHop Technologies when requesting information regarding operational issues or problems.

# Chapter 9
# Interacting with GateD

## 9.1 Overview

Certain actions performed by the system administrator while GateD is running can affect its operation. The kernel notifies applications of some state changes via the routing socket or **`ioctl()`** interfaces, depending on the operating system. The actions that can affect GateD include:

- Interface configuration (for example, **`ifconfig`**)
- Bringing interfaces up or down
- Adding or deleting interface aliases
- Routing table configuration (for example, **`route`**)
- Adding or deleting routes
- Kernel parameter configuration (for example, **`ndd`** or **`sysctl`**)
- Changing kernel parameter values (for example, IP-forwarding)

This document illustrates the way in which GateD will function in the presence of these types of changes. If configured to do so, GateD will write notifications to the global log file as it becomes aware of these types of changes.

The interface to the kernel (for example, **`ioctl()`** or a routing socket) affects the time delay between the time an event occurs and when GateD learns of it. The various kernel interfaces are discussed in “Forwarding Tables and Routing Tables” on page 96 in *Configuring GateD*.

## 9.2 Interface Configuration Changes

When an interface is changed to the 'down' state while GateD is running, all addresses on that interface are considered unusable. The GateD preference for the interface route is changed to the 'down' preference, which can be specified in the configuration file; see “Chapter 7 Interface Statement” on page 23 in *Configuring GateD* for more information. The default 'down' preference is 120.

When an interface is changed to the 'up' state while GateD is running, all addresses on that interface are considered usable. The interface route is changed to the 'direct' preference, which is very favorable (10) by default. This may also be changed in the **`interface`** statement in the configuration file.

When new aliases are added to an interface, a new logical address structure is allocated for the address and the protocols are notified. The address may be marked as a 'primary' address; see “Chapter 7 Interface Statement” on page 23 in *Configuring GateD* for more information on aliases.

The protocols are notified of these changes almost immediately after GateD learns this information from the kernel. Each configured protocol is notified separately of any changed interface state, including the netmask, aliases, and metric.

Unfortunately, most operating systems do not provide a way for GateD to learn of 'disconnected' interfaces, for example, when a cable is removed from a network adaptor.

## 9.3 Routing Table Changes

The system administrator should not change the state of the routing table while GateD is running. Some operating systems do not notify GateD of these types of changes, and the table may be left in an undesired state.

To add permanent routes to the table, use the **`static`** configuration option. (See “Chapter 19 Static Routes” on page 101 in *Configuring GateD* for more information.)

At startup, GateD reads the routing table and schedules non-interface routes for deletion as kernel remnants. The time delay for this action can be configured using the **`remnantholdtime`** option. (See “Chapter 18 Kernel Interface” on page 95 in *Configuring GateD*, for more information.)

## 9.4 Kernel Parameter Changes

At startup, GateD reads the values of some kernel configuration options. It should be noted that when these values have been changed, GateD is not notified by the kernel. It will be necessary to restart the daemon to learn of the new option values.

Also at startup, GateD attempts to find the maximum send and receive buffer sizes for each of its sockets via trial and error. The maximum sizes for some of these sockets can be set via a kernel configuration option. GateD is not notified by the kernel when these sizes have changed; changing them while the daemon is running may lead to undesirable results.